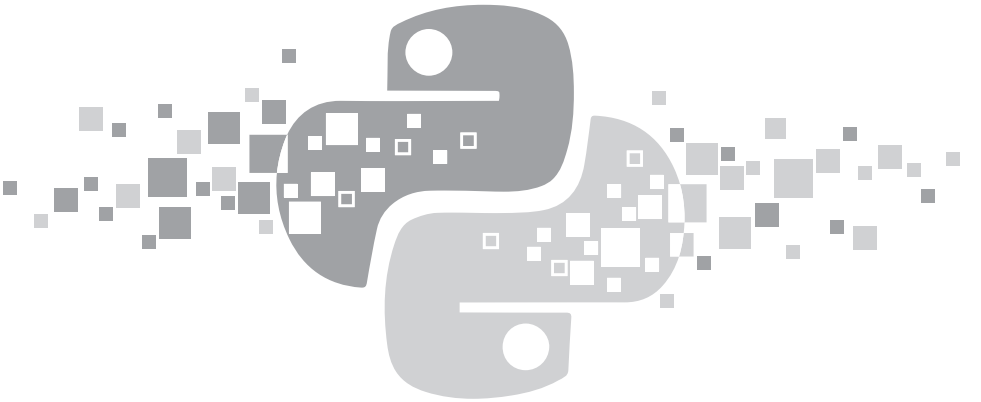
机器学习
Python实践

魏贞原 著



電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书系统地讲解了机器学习的基本知识，以及在实际项目中使用机器学习的基本步骤和方法；详细地介绍了在进行数据处理、分析时怎样选择合适的算法，以及建立模型并优化等方法，通过不同的例子展示了机器学习在具体项目中的应用和实践经验，是一本非常好的机器学习入门和实践的书籍。

不同于很多讲解机器学习的书籍，本书以实践为导向，使用 scikit-learn 作为编程框架，强调简单、快速地建立模型，解决实际项目问题。读者通过对本书的学习，可以迅速上手实践机器学习，并利用机器学习解决实际问题。本书非常适合于项目经理、有意从事机器学习开发的程序员，以及高校相关专业在的读学生阅读。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

机器学习：Python 实践 / 魏贞原著. —北京：电子工业出版社，2018.1

ISBN 978-7-121-33110-7

Ⅰ. ①机… Ⅱ. ①魏… Ⅲ. ①机器学习②软件工具－程序设计 Ⅳ. ①TP311.561②TP181

中国版本图书馆 CIP 数据核字（2017）第 288527 号

策划编辑：石　倩
责任编辑：徐津平
特约编辑：赵树刚
印　　刷：三河市双峰印刷装订有限公司
装　　订：三河市双峰印刷装订有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：787×980　1/16　　印张：14.25　　字数：251 千字
版　　次：2018 年 1 月第 1 版
印　　次：2018 年 1 月第 1 次印刷
定　　价：59.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件到 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819，faq@phei.com.cn。

序言

人工智能作为一种新技术，它的发展变迁可以用著名的 S 曲线来表示。具有划时代意义的新技术（Disruptive Innovation），往往出现在社会发展的成熟期。这时国家的 GDP 增长开始减缓，生产成本居高不下，相比之下社会需求却没有显著增加。现在的人工智能技术发展正处于这样一个时期，可以说是应运而热。但我们更关心的是它是否到了腾飞的前期。我个人觉得时机已经到来，原因有三：数字化、物联网及人工智能（AI）技术。

第一，社会生活的高度数字化进程已经让人与人之间的联结几乎完全可以用数字化的方式来描述。这一切极大地归功于智能手机和社交媒体的深度普及。关于我们的信息都被记录在信息系统里，无论是以格式化的形式（ERP、银行系统、电商系统等），还是以非格式化的形式（社交媒体上的文字或语音的交流），这些都已成为机器可以分析解读的数据，而且还在不断积累中。

第二，物联网技术使信息系统能够实时不间断地从我们的日常活动中获取新的信息，并且可以通过双向通信机制给予实时反馈。比如智能手表和智能扫地机器人等。

第三，人工智能技术本身的发展已经到了足以支撑大规模商业化应用的阶段。无论是人脸识别还是医疗文献分析，人工智能已经作为工具出现在我们的身边。

很多人还在争论人工智能是否会成为人类的敌人，尤其是在 AI 技术发展到

泛人工智能（Artificial General Intelligence，AGI）甚至超级人工智能（Artificial Super Intelligence，ASI）的时候。这是否会成为现实或者什么时候会成为现实谁也说不清楚。但在人机同行的今天，如果还不赶快学习充实自己，明天我们肯定会遇到已经用 AI 版本升级了的人类对手。这种痛也许只有百多年前抵抗英法联军的八旗子弟才领悟过，但为时已晚。如何不掉队，赶上甚至超越时代发展潮流，学习新技术是唯一手段。

就像前几次工业革命一样，人工智能带来的技术革命也有三个关键成功因素（3M）：数据（原材料：Raw Material）、技术（机器：Machine）及商业模式（Business Model）。在现实世界中，数据的金矿已经大量积累，并等待我们去开采和精炼。技术，如 Python、卷积神经网络等都已被成熟应用。商界精英更是想出了很多商业化的应用场景，在同声翻译、文本分析、医疗影像分析等领域展开了众多的投资和商业化应用。本书作为一本介绍 Python 的技术类专业书籍，立足机器学习中的监督式学习，围绕着课程、项目和方法深入浅出地介绍了如何使用 Python 来完成机器学习的相关工作。内容涵盖了人工智能的三个关键成功因素，以体系化和细致的讲解方便读者理解和学习有关机器学习的全过程。本书是一本实战性很强的参考书籍，帮助我们在人工智能时代迅速掌握新技术的精髓。

周德标
副合伙人
IBM 大中华区董事长执行助理
“周教授谈人工智能”微信公众号作者

前言

“这是最好的时代，也是最坏的时代”，这是英国文豪狄更斯的名著《双城记》开篇的第一句话，一百多年来不断被人引用。这里再次引用它来形容智能革命给我们带来的未来社会。从 2016 年 AlphaGo 在围棋比赛中战胜韩国选手李世石，到 2017 年 Master 战胜世界排名第一的围棋选手柯洁，人工智能再一次引起了世人的注意。在大数据出现之前，人工智能的概念虽然一直存在，但是计算机一直不擅长处理需要依赖人类的智慧解决的问题，现在换个思路就可以解决这些问题，其核心就是变智能问题为数据问题。由此，全世界开始了新一轮的技术革命——智能革命。

自从 1687 年艾萨克·牛顿发表了论文《自然定律》，对万有引力和三大运动定律进行了描述，人类社会进入了科学时代。在此之后，瓦特通过科学原理直接改进蒸汽机，开启了工业革命的篇章，由于机器的发明及运用成为这个时代的标志，因此历史学家称这个时代为“机器时代”。机器时代是利用机器代替人力，在原有的产业基础上加上蒸汽机形成新的产业，例如马车加上蒸汽机成为火车，改变了人的出行方式；帆船加上蒸汽机成为轮船，让货物的运输变得更加便捷。同时，原有的工匠被更加便宜的工人替代，社会的财富分配不均，社会进入动荡期，如英国大约花费了半个世纪的时间才完成了工业革命的变革。同样，第二次工业革命和信息革命，每一次变革都让财富更加集中，给社会带来动荡。第二次工业革命同样花费了半个世纪的时间，一代人才消除工业革命带来的影响，让大部分人受益。当前的智能革命也会带来财富的重新分配和社会的动荡，

当然目前的政府对这次革命的过程都有了足够的了解，能够把社会的动荡控制在最小范围，但是在变革中的人依然需要经受这次变革带来的动荡。

每一次变革都是一次思维方式的改进，工业革命是机器思维替代了农耕时代的思想；信息革命是香农博士（1916—2001 年）的信息论带来的思想方法替代机器思维，并成为社会主导思想；在这次智能革命中，以大数据为核心的思维方式将会主导这次变革。在历次的技术革命中，一个人、一家企业，甚至一个国家，可以选择的道路只有两条：要么加入变革的浪潮，成为前 2%的弄潮儿；要么观望徘徊，被淘汰。

要成为 2%的弄潮儿，需要积极拥抱这次智能变革，掌握在未来社会不会被淘汰的技能。在以大数据为基石的智能社会，利用机器学习算法对数据进行挖掘，是使机器更智能的关键，掌握数据挖掘是拥抱智能社会的举措之一。本书就将介绍如何利用机器学习算法来解决问题，对数据进行挖掘。

作　者

轻松注册成为博文视点社区用户（www.broadview.com.cn），扫码直达本书页面。

- **下载资源**：本书如提供示例代码及资源文件，均可在 下载资源 处下载。
- **提交勘误**：您对书中内容的修改意见可在 提交勘误 处提交，若被采纳，将获赠博文视点社区积分（在您购买电子书时，积分可用来抵扣相应金额）。
- **交流互动**：在页面下方 读者评论 处留下您的疑问或观点，与我们和其他读者一同学习交流。

页面入口：*http://www.broadview.com.cn/33110*

目录

第一部分　初始

第二部分 数据理解

第三部分　数据准备

第四部分 选择模型

第一部分

初始

像一个优秀的工程师一样使用机器学习，而不要像一个机器学习专家一样使用机器学习方法。

—— Google

1 初识机器学习

本书主要介绍机器学习在实践中的应用，介绍利用 Python 的生态环境，使用机器学习的算法来解决工程实践中的问题，而不是介绍算法本身。本书会通过例子一步一步地引导大家使用机器学习来处理和分类与回归模型相关的问题。

1.1 学习机器学习的误区

下面三点是利用 Python 进行机器学习的误区，应该尽量避免：

- 必须非常熟悉 Python 的语法和擅长 Python 的编程。
- 非常深入地学习和理解在 scikit-learn 中使用的机器学习的理论和算法。
- 避免或者很少参与完成项目，除机器学习之外的部分。

我相信这些方式对一部分人可能会非常有效，但是这会降低掌握机器学习技能的速度和要达到通过机器学习来解决问题的目标。这也会浪费大量时间单独学习机器学习算法，但却不知如何利用机器学习来解决现实中遇到的问题。

1.2 什么是机器学习

机器学习（Machine Learning，ML）是一门多领域的交叉学科，涉及概率论、统计学、线性代数、算法等多门学科。它专门研究计算机如何模拟和学习人的行为，以获取新的知识或技能，重新组织已有的知识结构使之不断完善自身的性能。

机器学习已经有了十分广泛的应用，例如：数据挖掘、计算机视觉、自然语言处理、生物特征识别、搜索引擎、医学诊断、检测信用卡欺诈、证券市场分析、DNA 序列测序、语音和手写识别、战略游戏和机器人运用。

机器学习的算法分为两大类：监督学习和无监督学习。

监督学习即在机器学习过程中提供对错指示。一般是在数据组中包含最终结果(0,1)，通过算法让机器自己减少误差。这一类学习主要应用于分类和预测（Regression & Classify）。监督学习从给定的训练数据集中学习出一个目标函数，当新的数据到来时，可以根据这个函数预测结果。监督学习的训练集要求包括输入和输出，也可以说包括特征和目标，训练集中的目标是由人标注的。常见的监督学习算法包括回归分析和统计分类。

非监督学习又称归纳性学习（Clustering），利用 K 方式（KMean）建立中心（Centriole），通过循环和递减运算（Iteration&Descent）来减小误差，达到分类的目的。

1.3 Python 中的机器学习

本书主要关注监督学习中的分类与回归问题处理的预测模型，这是在工业中应用非常广泛的分类，也是 scikit-learn 擅长的一个领域。与统计学不同，机器学习的预测模型是用来理解数据、解决问题的；聚焦于如何创建一个更加精准的模型，而不是用来解释模型是如何设置的。与大部分机器学习的领域不同的是，预测模型是使用表格格式的数据作为模型的输入的，因此数据的采集和整理是很重要的工作。

本书会围绕以下三部分来引导大家学习机器学习。

- **课程：**学习在项目中如何将机器学习的任务和 Python 有机地结合在一起，以便实现每一个机器学习问题的最佳实践。
- **项目：**通过实例来理解学到的预测模型的知识。
- **方法：**学到一系列方法，只是进行简单的复制粘贴操作就可以启动一个新的机器学习项目。

我们将通过项目来介绍基于 Python 的生态环境如何完成机器学习的相关工作。一旦明白了如何使用 Python 平台来完成机器学习的任务，就可以在不同的项目中重复使用这种方法解决问题。利用机器学习的预测模型来解决问题共有六个基本步骤，如图 1-1 所示。

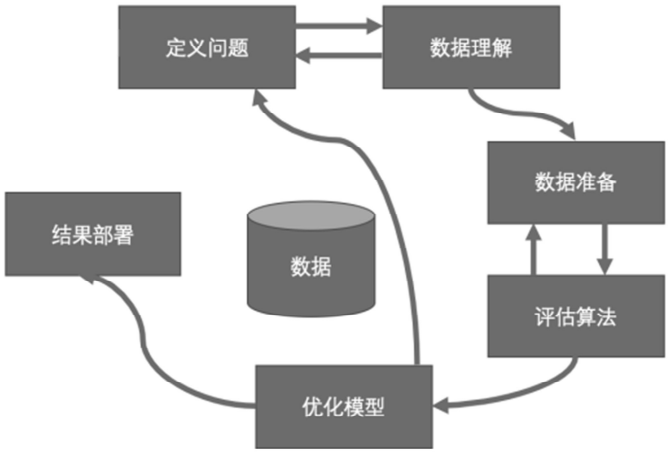


图 1-1

- **定义问题：**研究和提炼问题的特征，以帮助我们更好地理解项目的目标。
- **数据理解：**通过描述性统计和可视化来分析现有的数据。
- **数据准备：**对数据进行格式化，以便于构建一个预测模型。
- **评估算法：**通过一定的方法分离一部分数据，用来评估算法模型，并选取一部分代表数据进行分析，以改善模型。
- **优化模型：**通过调参和集成算法提升预测结果的准确度。
- **结果部署：**完成模型，并执行模型来预测结果和展示。

1.4 学习机器学习的原则

学习机器学习是一段旅程。需要知道自己具备的技能、目前所掌握的知识，以及明确要达到的目标。要实现自己的目标需要付出时间和辛勤的工作，但是在目标的实现过程中，有很多工具可以帮助你快速达成目标。

创建半正式的工作产品。以博客文章、技术报告和代码存储的形式记下学习和发现的内容，快速地为自己和他人提供一系列可以展示的技能、知识及反思。

实时学习。不能仅在需要的时候才学习复杂的主题，例如，应该实时学习足够的概率和线性代数的指示来帮助理解正在处理的算法。在开始进入机器学习领域之前，不需要花费太多的时间来专门学习统计和数学方面的知识，而是要在平时进行实时学习，积累知识。

利用现有的 Skills。如果可以编码，那么通过实现算法来理解它们，而不是研究数学理论。使用自己熟悉的编程语言，让自己专注于正在学习的一件事情上，不要同时学习一种新的语言、工具或类库，这样会使学习过程复杂化。

掌握是理想。掌握机器学习需要持续不断的学习。也许你永远不可能实现掌握机器学习的目标，只能持续不断地学习和改进所掌握的知识。

1.5 学习机器学习的技巧

下面三个技巧可以有效地帮助你快速提高学习机器学习的能力。

- 启动一个可以在一个小时内完成的小项目。
- 通过每周完成一个项目来保持你的学习势头，并建立积累自己的项目工作区。
- 在微博、微信、Github 等社交工具上分享自己的成果，或者随时随地地展示自己的兴趣，增加技能、知识，并获得反馈。

1.6 这本书不涵盖以下内容

这本书是写给对机器学习感兴趣和立志学习机器学习的 Python 程序员的，是一本关于机器学习实践的书籍。

这不是一本关于机器学习的教科书。本书只会简单介绍机器学习的基本原理和算法。在这里假设你已经掌握了机器学习的基础知识，或者有能力自己来完成机器学习的基础知识的学习。

这不是一本算法书。本书不会详细介绍机器学习的算法。在这里假设你已经掌握了机器学习的相关算法，或者能够独立完成相关算法知识的学习。

这不是一本关于 Python 的语法书。本书不会花费大量的篇幅来讲解 Python 的语法。在这里假设你是一个经验丰富的开发人员，能够快速掌握一种类似于 C 语言的开发语言。

1.7 代码说明

本书中的所有代码都在 Python3.6 中调试过，可以下载这些代码来加快学习的进度。不过笔者还是强烈建议读者独立完成每一行代码，以加深理解。所有的源代码和数据集都保存在 Github 上，可以自行下载，地址为：https://github.com/weizy1981/MachineLearning。

1.8 总结

本章已经对本书的内容做了一个概要的介绍。这本书与其他机器学习书籍的不同之处在于不会花费大量的篇幅来讲解算法，而是通过构建一个预测模型的机器学习项目，来引导学习机器学习在实践中的应用，并在项目实践中掌握机器学习知识。

下一章会介绍 Python 在机器学习方面的生态环境，并分析 Python 和 SciPy 在机器学习方面的优势，以及如何使用这些平台。

2 Python 机器学习的生态圈

随着 Python 生态圈的发展，在机器学习领域，Python 已经发展成为机器学习方面最主要的语言。Python 为什么能够成为机器学习的主流语言？这是因为 Python 不仅可以应用在 R&D 科研部门，也可以应用在实际的生产当中。本章主要介绍 Python 在机器学习方面的生态圈。完成本章后将会学到：

- Python 和它在日益改进的机器学习中的应用。
- SciPy 和它提供的基于 NumPy、Matplotlib 及 Pandas 的功能。
- scikit-learn 提供的机器学习的算法。
- 如何设置基于 Python 的机器学习的环境。

2.1 Python

Python 是一门面向对象的动态解释语言，简单易学，并且具有很好的可读性。Python 语法简洁清晰，特色之一是强制用空白符（White Space）作为语句缩进。Python 具有丰富和强大的类库，它常被称为“胶水语言”，能够很轻松地把用其他语言制作的各种模块（尤其是 C/C++）联结在一起。常见的一种应用情形是，先使用 Python 快速生成程序的

原型（有时甚至是程序的最终界面），然后对其中有特别要求的部分用更合适的语言改写，比如 3D 游戏对图形渲染模块的性能要求特别高，就可以用 C/C++重写，而后封装为Python可以调用的扩展类库。目前，在Python的生态圈中存在大量的第三方扩展类库，可以借助这些类库轻松实现项目需求。需要注意的是，在使用扩展类库时可能需要考虑平台问题，某些扩展类库可能不提供跨平台的实现。Python 目前是一门非常流行的语言，在 TIOBE 2017 年 6 月编程语言排名中占据第四名的位置，如图 2-1 所示。

Jun 2017	Jun 2016	Change	Programming Language	Ratings	Change
1	1		Java	14.493%	-6.30%
2	2		C	6.848%	-5.53%
3	3		C++	5.723%	-0.48%
4	4		Python	4.333%	+0.43%
5	5		C#	3.530%	-0.26%

图 2-1

在 PYPL 各种编程语言的流行程度的统计中，2017 年 6 月 Python 排在第二名，如图 2-2 所示。

Worldwide, Jun 2017 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Java	22.7 %	-1.2 %
2		Python	16.1 %	+3.8 %
3		PHP	9.3 %	-0.9 %
4		C#	8.2 %	-0.6 %
5		Javascript	7.9 %	+0.5 %

图 2-2

Python 是一门动态语言，非常适合于交互性开发和大型项目的快速原型开发。由于 Python 具有丰富的类库支持，因此被广泛应用于机器学习和数据科学方面。从这个方面来说，利用 Python 可以将研究项目和生产项目用统一的语言来实现，这就有效地降低了将研究项目转化成生产项目的成本。

2.2 SciPy

SciPy 是在数学运算、科学和工程学方面被广泛应用的 Python 类库。它包括统计、优化、整合、线性代数模块、傅里叶变换、信号和图像处理、常微分方程求解器等，因此被广泛地应用在机器学习项目中。SciPy 依赖以下几个与机器学习相关的类库。

NumPy：是 Python 的一种开源数值计算扩展。它可用来存储和处理大型矩阵，提供了许多高级的数值编程工具，如矩阵数据类型、矢量处理、精密的运算库。

Matplotlib：Python 中最著名的 2D 绘图库，十分适合交互式地进行制图；也可以方便地将它作为绘图控件，嵌入 GUI 应用程序中。

Pandas：是基于 NumPy 的一种工具，是为了解决数据分析任务而创建的。Pandas 纳入了大量库和一些标准的数据模型，提供了高效地操作大型数据集所需的工具，也提供了大量能使我们快速、便捷地处理数据的函数和方法。

安装并熟悉 SciPy 是提高机器学习实践的有效手段，尤其是在以下几个方面：

- 可以利用 NumPy 数组来准备机器学习算法的数据。
- 可以使用 Matplotlib 来创建图表，展示数据。
- 通过 Pandas 导入、展示数据，以便增强对数据的理解和数据清洗、转换等工作。

2.3 scikit-learn

scikit-learn 是 Python 中开发和实践机器学习的著名类库之一，依赖于 SciPy 及其相关类库来运行。scikit-learn 的基本功能主要分为六大部分：分类、回归、聚类、数据降维、模型选择和数据预处理。需要指出的是，由于 scikit-learn 本身不支持深度学习，也不支持 GPU 加速，因此 scikit-learn 对于多层感知器（MLP）神经网络的实现并不适合处理大规模问题。（scikit-learn 对 MLP 的支持在 0.18 版之后增加）

scikit-learn 是一个开源项目，遵守 BSD 协议，可以将项目应用于商业开发。目前主要由社区成员自发进行维护。可能是由于维护成本的限制，scikit-learn 相比其他项目要

显得更为保守，这主要体现在两个方面：

- scikit-learn 从来不做除机器学习领域之外的其他扩展。
- scikit-learn 从来不采用未经广泛验证的算法。

2.4 环境安装

前面介绍了在本书中使用的相关机器学习的类库，接下来将介绍如何进行安装，有多种方法可以用来配置机器学习的相关环境。在安装之前先说明一下，笔者使用的 IDE 是 PyCharm，本书中相关示例的演示都基于 PyCharm 进行。PyCharm 是 JetBrains 公司的产品，有收费的专业版和免费的社区版，在本书的学习过程中使用社区版就可以满足需求。可以自行到 JetBrains 的官网（http://www.jetbrains.com/pycharm/）进行下载。具体的 PyCharm 的说明见附录 A.1。

2.4.1 安装 Python

可以在 python.org（https://www.python.org）下载合适的版本，并参照相应的安装说明进行安装，推荐安装 Python3.6.1，本书中的所有示例都将基于 Python3.6 进行。当安装完成后，在命令终端执行以下命令来确认安装版本。

```
python3 --version
```

看到如下的实现后说明安装已经完成。

```
Python 3.6.1
```

2.4.2 安装 SciPy

有多种方法可以用于安装 SciPy，本书推荐使用 Python 的安装包管理工具 pip 进行安装。SciPy 的文档非常完善，覆盖了多个平台和多种安装方式，请参考 SciPy 的安装指南（https://www.scipy.org/install.html）进行安装。在安装 SciPy 时，请确保至少安装了 SciPy、Numpy、Matplotlib 和 Pandas。

安装完成后，可以在 IDE 里新建一个 Python 文件，输入以下代码并执行，确认安装结果。

```
import scipy
import numpy
import matplotlib
import pandas
print('scipy:{}'.format(scipy.__version__))
print('numpy:{}'.format(numpy.__version__))
print('matplotlib:{}'.format(matplotlib.__version__))
print('pandas:{}'.format(pandas.__version__))
```

在我的机器上的执行结果如下：

```
scipy:0.19.0
numpy:1.13.0
matplotlib:2.0.2
pandas:0.20.2
```

2.4.3 安装 scikit-learn

建议采用与安装 SciPy 相同的方法来安装 scikit-learn，通过 pip 来管理安装包。安装完成后，可以通过相似的方法来确认安装结果。代码如下：

```
import sklearn
print('sklearn:{}'.format(sklearn.__version__))
```

在我的机器上的执行结果如下：

```
sklearn:0.18.1
```

2.4.4 更加便捷的安装方式

如果觉得通过 pip 管理安装包比较复杂，还可以利用一个更加简单便捷的方式来安装这些安装包：通过 Anaconda 安装相关的安装包。Anaconda 提供了包管理与环境管理的功能，可以很方便地解决多版本 Python 并存、切换及各种第三方包的安装问题。Anaconda 是免费的，可以自由地进行下载（https://www.continuum.io/downloads）和安装，目前支持 Windows、Mac OS 和 Linux 系统，读者可以选择适合自己的操作系统下载安装。

2.5 总结

本章主要介绍了 Python 及其在机器学习方面的生态圈和相关类库的安装，包括以下内容：

- Python 及其在机器学习方面的类库和应用。
- SciPy 的主要功能和它依赖的扩展类库。
- scikit-learn 及它提供的机器学习算法。

接下来将介绍一个机器学习的实例，这个实例也被称为机器学习中的 Hello World。通过这个实例可以使读者对机器学习的项目有一个初步的了解，了解机器学习项目的基本步骤和流程。

3 第一个机器学习项目

前面介绍了在 Python 中进行机器学习实践需要的生态环境，接下来将会通过鸢尾花分类这个例子对机器学习做一个概要的介绍。本章通过一步一步地实现这个项目来介绍以下内容。

- 下载和安装在 Python 中机器学习的各个方面的类库。
- 导入数据，并通过描述性分析、可视化等对数据进行分析。
- 创建六个模型，并从中选择准确度最高的模型。

这一章介绍的项目是机器学习中的Hello World项目，是给初学者设计的一个指南项目。

3.1 机器学习中的 Hello World 项目

这个项目是针对鸢尾花（Iris Flower）进行分类的一个项目，数据集是含鸢尾花的三个亚属的分类信息，通过机器学习算法生成一个模型，自动分类新数据到这三个亚属的某一个中。项目中使用的鸢尾花数据集是一个非常容易理解的数据集，这个数据集具有以下特点：

- 所有的特征数据都是数字，不需要考虑如何导入和处理数据。
- 这是一个分类问题，可以很方便地通过有监督学习算法来解决问题。
- 这是一个多分类问题，也许需要一些特殊的处理。
- 所有的特征的数值采用相同的单位，不需要进行尺度的转换。

接下来我们将通过这个例子一步步地来展示一个机器学习项目的所有步骤。我们将按照下面的步骤实现这个项目：

（1）导入数据。

（2）概述数据。

（3）数据可视化。

（4）评估算法。

（5）实施预测。

我们需要认真完成每一步，尝试自己输入每一行代码，以加深对机器学习项目流程的理解。请启动你的 Python 环境或 IDE，开始实现机器学习的第一个项目吧。

3.2 导入数据

导入项目所需的类库和鸢尾花（Iris Flower）数据集。

3.2.1 导入类库

导入在项目中将要使用的类库和方法。代码如下：

```
# 导入类库
from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

所有类库的导入都不应有错误提示。如果出现了错误提示，那么暂时停下来，先设置一个能够运行的 SciPy 环境。

3.2.2 导入数据集

我们可以在 UCI 机器学习仓库下载鸢尾花（Iris Flower）数据集（http://archive.ics.uci.edu/ml/datasets/Iris），下载完成后保存在项目的统计目录中。在这里将使用 Pandas 来导入数据和对数据进行描述性统计分析，并利用 Matplotlib 实现数据可视化。需要注意的是，在导入数据时，为每个数据特征设定了名称，这有助于后面对数据的展示工作，尤其是通过图表展示数据。代码如下：

```
# 导入数据
filename = 'iris.data.csv'
names = ['separ-length', 'separ-width', 'petal-length', 'petal-width',
'class']
dataset = read_csv(filename, names=names)
```

3.3 概述数据

我们需要先看一下数据，增加对数据的理解，以便选择合适的算法。我们将从以下几个角度来审查数据：

（1）数据的维度。

（2）查看数据自身。

（3）统计描述所有的数据特征。

（4）数据分类的分布情况。

不要担心这会需要很多代码，每一种审查方法只有一行代码。这些代码非常有效，在以后的项目中也会用到。

3.3.1 数据维度

通过查看数据的维度，可以对数据集有一个大概的了解，如数据集中有多少行数据、数据有几个属性等。代码如下：

```
#显示数据维度
print('数据维度：行 %s，列 %s' % dataset.shape)
```

将会得到一个具有 150 行数据，5 个数据特征属性的结果，执行结果如下：

```
数据维度：行 150，列 5
```

3.3.2 查看数据自身

查看数据自身也是一个很好的理解数据的方法，通过查看数据可以直观地看到数据的特征、数据的类型，以及大概的数据分布范围等。代码如下：

```
# 查看数据的前 10 行
print(dataset.head(10))
```

在这里查看前 10 行记录，执行结果如下：

```
   separ-length  separ-width  petal-length  petal-width  class
0           5.1          3.5           1.4          0.2   Iris-setosa
1           4.9          3.0           1.4          0.2   Iris-setosa
2           4.7          3.2           1.3          0.2   Iris-setosa
3           4.6          3.1           1.5          0.2   Iris-setosa
4           5.0          3.6           1.4          0.2   Iris-setosa
5           5.4          3.9           1.7          0.4   Iris-setosa
6           4.6          3.4           1.4          0.3   Iris-setosa
7           5.0          3.4           1.5          0.2   Iris-setosa
```

```
8           4.4         2.9         1.4         0.2    Iris-setosa
9           4.9         3.1         1.5         0.1    Iris-setosa
```

3.3.3 统计描述数据

数据特征的统计描述信息包括数据的行数、中位值、最大值、最小值、均值、四分位值等统计数据信息。代码如下：

```
# 统计描述数据信息
print(dataset.describe())
```

执行结果如下：

```
       separar-length  separar-width  petal-length  petal-width
count      150.000000     150.000000    150.000000    150.000000
mean         5.843333       3.054000      3.758667      1.198667
std          0.828066       0.433594      1.764420      0.763161
min          4.300000       2.000000      1.000000      0.100000
25%          5.100000       2.800000      1.600000      0.300000
50%          5.800000       3.000000      4.350000      1.300000
75%          6.400000       3.300000      5.100000      1.800000
max          7.900000       4.400000      6.900000      2.500000
```

3.3.4 数据分类分布

接下来看一下数据在不同分类的分布情况，执行程序后得到的将是每个分类数据量的绝对的数值，看一下各个分类的数据分布是否均衡。代码如下：

```
# 分类分布情况
print(dataset.groupby('class').size())
```

这里就是通过前面设定的数据特征名称来查看数据的。执行结果如下：

```
class
Iris-setosa       50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

我们可以看到鸢尾花的三个亚属的数据各 50 条，分布非常平衡。如果数据的分布不平衡时，可能会影响到模型的准确度。因此，当数据分布不平衡时，需要对数据进行处理，调整数据到相对平衡的状态。调整数据平衡时有以下几种方法。

- **扩大数据样本**：这是一个容易被忽视的选择。一个更大的数据集，就有可能挖掘出不同的或许更平衡的方面提高算法模型的准确度。
- **数据的重新抽样**：过抽样（复制少数类样本）和欠抽样（删除多数类样本）。当数据量很大时可以考虑测试欠抽样（大于一万条记录），当数据量比较少时可以考虑过抽样。
- **尝试生成人工样本**：一种简单的生成人工样本的方法是从少数类的实例中随机抽样特征属性，生成更多的数据。
- **异常检测和变化检测**：尝试用不同的观点进行思考，以解决问题。异常检测是对罕见事件的检测。这种思维的转变在于考虑以小类作为异常值类，它可以帮助获得一种新方法来分离和分类样本。

3.4 数据可视化

通过对数据集的审查，对数据有了一个基本的了解，接下来将通过图表来进一步查看数据特征的分布情况和数据不同特征之间的相互关系。

- 使用单变量图表可以更好地理解每一个特征属性。
- 多变量图表用于理解不同特征属性之间的关系。

3.4.1 单变量图表

单变量图表可以显示每一个单独的特征属性，因为每个特征属性都是数字，因此我们可以通过箱线图来展示属性与中位值的离散速度。代码如下：

```
 # 箱线图
dataset.plot(kind='box',   subplots=True,   layout=(2,2),   sharex=False,
sharey=False)
pyplot.show()
```

执行结果如图 3-1 所示。

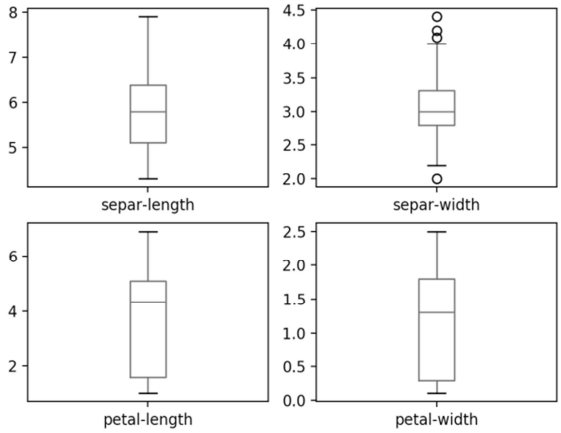


图 3-1

还可以通过直方图来显示每个特征属性的分布状况。代码如下：

```
# 直方图
dataset.hist()
pyplot.show()
```

在输出的图表中，我们看到 separ-length 和 separ-width 符合高斯分布。执行结果如图 3-2 所示。

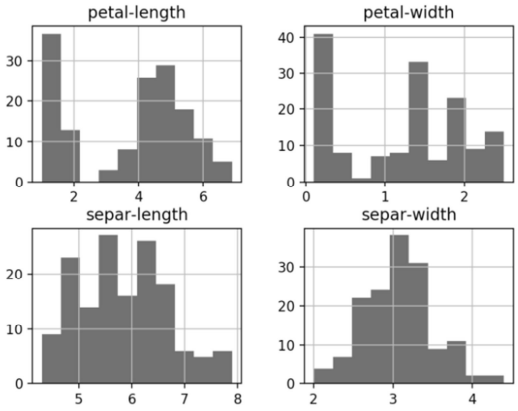


图 3-2

3.4.2 多变量图表

通过多变量图表可以查看不同特征属性之间的关系。我们通过散点矩阵图来查看每个属性之间的影响关系。

```
# 散点矩阵图
scatter_matrix(dataset)
pyplot.show()
```

执行结果如图 3-3 所示。

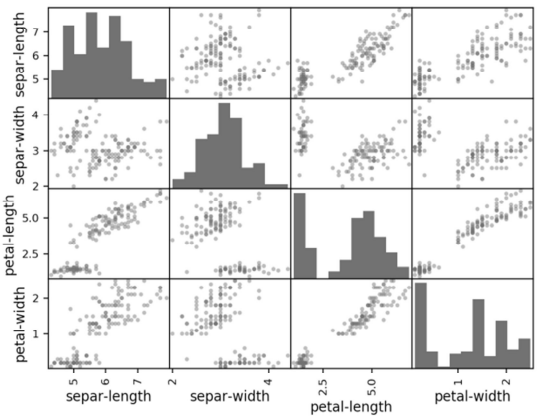


图 3-3

3.5 评估算法

通过不同的算法来创建模型，并评估它们的准确度，以便找到最合适的算法。将按照下面的步骤进行操作：

（1）分离出评估数据集。

（2）采用 10 折交叉验证来评估算法模型。

（3）生成 6 个不同的模型来预测新数据。

（4）选择最优模型。

3.5.1 分离出评估数据集

模型被创建后需要知道创建的模型是否足够好。在选择算法的过程中会采用统计学方法来评估算法模型。但是，我们更想知道算法模型对真实数据的准确度如何，这就是保留一部分数据来评估算法模型的主要原因。下面将按照 80%的训练数据集，20%的评估数据集来分离数据。代码如下：

```
# 分离数据集
array = dataset.values
X = array[:, 0:4]
Y = array[:, 4]
validation_size = 0.2
seed = 7
X_train, X_validation, Y_train, Y_validation = \
    train_test_split(X, Y, test_size=validation_size,
    random_state=seed)
```

现在就分离出了 X_train 和 Y_train 用来训练算法创建模型，X_validation 和 Y_validation 在后面会用来验证评估模型。

3.5.2 评估模式

在这里将通过 10 折交叉验证来分离训练数据集，并评估算法模型的准确度。10 折交叉验证是随机地将数据分成 10 份：9 份用来训练模型，1 份用来评估算法。后面我们会使用相同的数据对每一种算法进行训练和评估，并从中选择最好的模型。

3.5.3 创建模型

对任何问题来说，不能仅通过对数据进行审查，就判断出哪个算法最有效。通过前面的图表，发现有些数据特征符合线性分布，所有可以期待算法会得到比较好的结果。接下来评估六种不同的算法：

- 线性回归（LR）。
- 线性判别分析（LDA）。

- K 近邻（KNN）。
- 分类与回归树（CART）。
- 贝叶斯分类器（NB）。
- 支持向量机（SVM）。

这个算法列表中包含了线性算法（LR 和 LDA）和非线性算法（KNN、CART、NB 和 SVM）。在每次对算法进行评估前都会重新设置随机数的种子，以确保每次对算法的评估都使用相同的数据集，保证算法评估的准确性。接下来就创建并评估这六种算法模型。代码如下：

```
 # 算法审查
models = {}
models['LR'] = LogisticRegression()
models['LDA'] = LinearDiscriminantAnalysis()
models['KNN'] = KNeighborsClassifier()
models['CART'] = DecisionTreeClassifier()
models['NB'] = GaussianNB()
models['SVM'] = SVC()
# 评估算法
results = []
for key in models:
    kfold = KFold(n_splits=10, random_state=seed)
    cv_results = cross_val_score(models[key], X_train,
 Y_train, cv=kfold, scoring='accuracy')
    results.append(cv_results)
    print('%s: %f (%f)' %(key, cv_results.mean(),
 cv_results.std()))
```

3.5.4 选择最优模型

现在已经有了六种模型，并且评估了它们的精确度。接下来就需要比较这六种模型，并选出准确度最高的算法。执行上面的代码，结果如下：

```
LR: 0.966667 (0.040825)
LDA: 0.975000 (0.038188)
KNN: 0.983333 (0.033333)
CART: 0.975000 (0.038188)
```

```
NB: 0.975000 (0.053359)
SVM: 0.991667 (0.025000)
```

在这六种算法中有一个分类与回归树算法（CART），树算法生成的树模型可以通过图像的方式展示出来，树模型的可视化请参考附录 A。

通过上面的结果，很容易看出 SVM 算法具有最高的准确度得分。接下来创建一个箱线图，通过图表来比较算法的评估结果。代码如下：

```
# 箱线图比较算法
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(models.keys())
pyplot.show()
```

执行结果如图 3-4 所示。

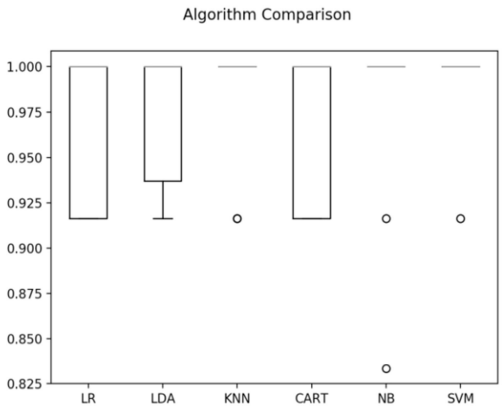


图 3-4

3.6 实施预测

评估的结果显示，支持向量机（SVM）是准确度最高的算法。现在使用预留的评估数据集来验证这个算法模型。这将会对生成的算法模型的准确度有一个更加直观的认识。

现在使用全部训练集的数据生成支持向量机（SVM）的算法模型，并用预留的评估数据集给出一个算法模型的报告。代码如下：

```
#使用评估数据集评估算法
svm = SVC()
svm.fit(X=X_train, y=Y_train)
predictions = svm.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

执行程序后，看到算法模型的准确度是 0.93。通过冲突矩阵看到只有两个数据预测错误。最后还提供了一个包含精确率（precision）、召回率（recall）、F1 值（F1-score）等数据的报告。结果如下：

```
0.933333333333
[[ 7  0  0]
 [ 0 10  2]
 [ 0  0 11]]
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       1.00      0.83      0.91        12
 Iris-virginica       0.85      1.00      0.92        11

    avg / total       0.94      0.93      0.93        30
```

3.7 总结

到这里已经完成了第一个机器学习项目。这个项目包含从数据导入到生成模型，以及通过模型对数据进行分类的全部过程。通过这个项目我们熟悉了机器学习项目中的各个步骤。接下来将介绍在本章中用到的对数据进行处理分析的技巧和算法，以及改进算法的实践。

4

Python 和 SciPy 速成

利用 Python 的资源进行机器学习方面的开发与实践，不需要成为一个专业的 Python 开发人员，要学习并理解 Python 的特征，并将已理解掌握的其他语言的技能转换到 Python 上即可。一个掌握了一门或多门语言的开发人员，可以很快地掌握 Python 语言。通过本章你将：

- 熟悉 Python 的基本语法。
- 熟悉 NumPy、Matplotlib 和 Pandas，让你能够阅读和编写基于 Python 的机器学习的脚本。
- 为深入地理解机器学习打下基础。

如果你已经对 Python 有了一些了解，学习这一章将会让你觉得非常轻松。

4.1 Python 速成

在开始使用 Python 之前，需要掌握几个 Python 语法，以便能够阅读和理解 Python 代码。

- 基本数据类型和赋值运算。

- 控制语句。
- 复杂数据类型。
- 函数。

接下来会通过一些例子来讲解上述内容。你也可以编写和测试这些内容来加强理解。在这里提醒大家一点，在 Python 中，空格是有意义的，用来区分代码块。

4.1.1 基本数据类型和赋值运算

作为一个程序员，对数据类型和赋值运算应该非常熟悉。下面简单地通过几个例子介绍一下 Python 的基本数据类型和赋值运算。

字符串

字符串的简单例子，代码如下：

```
# 字符串
data = 'Hello, world!'
print(data[0])
print(data[1:5])
print(len(data))
print(data)
```

在这里要注意 print(data[0])和 print(data[1:5])，这是字符串操作中对单个字符或指定范围的字符访问的方式。执行结果如下：

```
H
ello
13
Hello, world!
```

数值

数值的简单例子，代码如下：

```
# 数值
value = 523
print(value)
```

```
value = 6.18
print(value)
```

执行结果如下：

```
523
6.18
```

布尔类型

布尔类型的简单例子，代码如下：

```
# 布尔类型
true = True
false = False
print(true)
print(false)
```

执行结果如下：

```
True
False
```

多变量赋值

Python 中的多赋值运算，代码如下：

```
# 多变量赋值运算
a, b, c =1,'hello',True
print(a, b, c)
print(a)
print(b)
print(c)
```

从程序的可读性上来看，建议不使用或尽量少使用多变量赋值运算。执行结果如下：

```
1 hello True
1
hello
True
```

空值

在 Python 中，每一种数据类型都是对象，空值是 Python 中的一个特殊值，用 None 表示，表示该值是一个空对象。你可以将 None 赋值给任何变量，也可以将任何变量赋值给一个 None 值的对象。代码如下：

```
# 空值
a = None
b = a
print(a)
print(b)
```

执行结果如下：

```
None
None
```

4.1.2 控制语句

在 Python 中，控制语句主要有三类：条件控制语句、循环语句或条件循环语句。

条件控制语句

条件控制语句的简单例子，代码如下：

```
# 条件控制语句
value = 1
if value == 1:
    print('This is ture')
elif value > 10:
    print('it is bigger than 10? Yes it is true.')
else:
    print('This is false')
```

执行结果如下：

```
This is ture
```

在这里需要注意，每一个条件判断都以冒号（:）结尾，并且通过空格缩进来区分代码块。

循环语句

循环语句打印输出的简单例子，代码如下：

```
# 循环语句
for i in range(5):
    print(i)
```

执行结果如下：

```
0
1
2
3
4
```

条件循环语句

条件循环语句输出的简单例子，代码如下：

```
# 条件循环语句
i =0
while i < 5:
    print(i)
    i = i +1
```

执行结果如下：

```
0
1
2
3
4
```

4.1.3 复杂数据类型

在 Python 中有三种数据类型非常有用，并且会被经常使用到。它们分别是元组、列表和字典，列表和字典更是经常被使用到。

元组

元组是一个只读的集合类型，初始化后，元组的元素不能重新赋值。代码如下：

```
# 元组
a = (1, 2, 3)
print(a)
print(a[1])
```

执行结果如下：

```
(1, 2, 3)
2
```

列表

列表与元组类似，只是列表通过中括号定义，而且列表的元素可以重新赋值。对列表增加列表项使用列表的 append()函数。代码如下：

```
# 列表
a = [1, 2, 3]
print(a)
# 增加列表项
a.append(4)
print(a)
print(a[3])
# 更新列表项
a[2] = 5
print(a)
for i in a:
    print(i)
```

执行结果如下：

```
[1, 2, 3]
[1, 2, 3, 4]
4
[1, 2, 5, 4]
1
2
```

```
5
4
```

字典

字典是另一种可变容器模型，且可存储任意类型的对象。字典的每个键值对（key,value）用冒号（:）分隔，每个键值对之间用逗号（,）分隔，整个字典包括在花括号（{}）中。代码如下：

```
# 字典
mydict = {'a': 6.18, 'b': 'str', 'c': True}
print('A value: %.2f' % mydict['a'])
# 增加字典元素
mydict['a'] = 523
print('A value: %d' % mydict['a'])
print('keys: %s' % mydict.keys())
print('values: %s' % mydict.values())
for key in mydict:
    print(mydict[key])
```

执行结果如下：

```
A value: 6.18
A value: 523
keys: dict_keys(['a', 'b', 'c'])
values: dict_values([523, 'str', True])
523
str
True
```

此外，若删除字典中的全部元素，使用字典自身的 clear()方法；若删除字典特定的 key 元素，用 pop(key)方法。

```
mydict = {'a': 6.18, 'b': 'str', 'c': True}
# 删除特定元素
mydict.pop('a')
print(mydict)
# 删除字典中的全部元素
```

```
mydict.clear()
print(mydict)
```

执行结果如下：

```
{'b': 'str', 'c': True}
{}
```

4.1.4 函数

函数是组织好的、可重复使用的、用来实现单一或相关联功能的代码段。函数能提高应用的模块性和代码的重复利用率。目前我们已经使用了 Python 提供的许多内建函数，如 print()。而且，也可以自己创建函数，即自定义函数。

自定义函数需要遵循以下几个简单的规则：

- 函数代码块以 def 关键词开头，后接函数标识符名称和圆括号()。
- 任何传入参数和自变量必须放在圆括号中间，圆括号之间可以用于定义参数。
- 函数的第一行语句可以选择性地使用文档字符串，用于存放函数说明。
- 函数内容以冒号起始，并且缩进。
- 用 return [表达式] 结束函数，选择性地返回一个值给调用方。不带表达式的 return 相当于返回 None。

下面是一个求和的自定义函数的示例，代码如下：

```
# 定义函数
def mysum(x, y):
    "这是自定义函数"
    return x + y
# 测试函数
result = mysum(x=1, y=2)
print(result)
```

执行结果如下：

```
3
```

4.1.5 with 语句

要使用 with 语句，首先要明白上下文管理器这一概念。有了上下文管理器，with 语句才能工作。

下面是一组与上下文管理器和 with 语句有关的概念。

上下文管理协议（Context Management Protocol）：包含方法__enter__() 和__exit__()，支持该协议的对象要实现这两个方法。

上下文管理器（Context Manager）：支持上下文管理协议的对象，这种对象实现了__enter__() 和__exit__() 方法。上下文管理器定义执行 with 语句时要建立的运行时上下文，负责执行 with 语句块上下文中的进入与退出操作。通常使用 with 语句调用上下文管理器，也可以通过直接调用其方法来使用。

运行时上下文（Runtime Context）：由上下文管理器创建，通过上下文管理器的__enter__() 和__exit__() 方法实现。__enter__() 方法在语句体执行之前进入运行时上下文，__exit__() 方法在语句体执行完毕后从运行时上下文退出。with 语句支持运行时上下文这一概念。

上下文表达式（Context Expression）：with 语句中跟在关键字 with 之后的表达式，该表达式要返回一个上下文管理器对象。

语句体（with-body）：with 语句包裹起来的代码块，在执行语句体之前会调用上下文管理器的__enter__()方法，执行完语句体之后会执行__exit__()方法。

Python 对一些内建对象进行改进，加入了对上下文管理器的支持，可以用于 with 语句中，比如可以自动关闭文件、线程锁的自动获取和释放等。假设要对一个文件进行操作，可以使用 with 语句，代码如下：

```
with open('somefileName') as somefile:
    for line in somefile:
       print(line)
       # ...more code
```

这里使用了 with 语句，不管在处理文件过程中是否发生异常，都能保证 with 语句执行完毕后关闭了打开的文件句柄。如果使用传统的 try/finally 范式，则要使用如下代码：

```
somefile = open('somefileName')
try:
   for line in somefile:
      print(line)
      # ...more code
finally:
   somefile.close()
```

使用 with 语句，简化了对异常的处理。因此，当需要对异常进行处理时，如果对象遵循了上下文管理协议（Context Management Protocol），建议使用 with 语句来实现。

4.2 NumPy 速成

NumPy 为 SciPy 提供了基本的数据结构和运算，其中最主要的是 ndarrays 多维数组，它提供了高效的矢量运算功能。

4.2.1 创建数组

利用 NumPy 创建多维数组非常简单，通过给 array 函数传递 Python 的序列对象创建数组，如果传递的是多层嵌套的序列，将创建多维数组。代码如下：

```
import numpy as np
# 创建数组
myarray = np.array([1, 2, 3])
print(myarray)
print(myarray.shape)
#创建多维数组
myarray = np.array([[1, 2, 3], [2, 3, 4], [3, 4, 5]])
print(myarray)
print(myarray.shape)
```

执行结果如下：

```
[1 2 3]
(3,)
[[1 2 3]
[2 3 4]
[3 4 5]]
(3, 3)
```

4.2.2 访问数据

对于 ndarray 数组的访问，我们可以通过数组的下标访问某一行，也可以访问某一列。代码如下：

```
import numpy as np
# 创建多维数组
myarray = np.array([[1, 2, 3], [2, 3, 4], [3, 4, 5]])
print(myarray)
print(myarray.shape)
# 访问数据
print('这是第一行：%s' % myarray[0])
print('这是最后一行：%s' % myarray[-1])
print('访问整列（3 列）数据：%s' % myarray[:, 2])
print('访问指定行（2 行）和列（3 列）的数据：%s' % myarray[1, 2])
```

执行结果如下：

```
[[1 2 3]
[2 3 4]
[3 4 5]]
(3, 3)
这是第一行：[1 2 3]
这是最后一行：[3 4 5]
访问整列（3 列）数据：[3 4 5]
访问指定行（2 行）和列（3 列）的数据：4
```

4.2.3 算数运算

使用 NumPy 的 ndarray 数组可以直接进行算术运算，或者说向量运算。代码如下：

```
import numpy as np
# 创建多维数组
myarray1 = np.array([[1, 2, 3], [2, 3, 4], [3, 4, 5]])
myarray2 = np.array([[11, 21, 31], [21, 31, 41], [31, 41, 51]])
print('向量加法运算：')
print(myarray1 + myarray2)
print('向量乘法运算：')
print(myarray1 * myarray2)
```

执行结果如下：

```
向量加法运算：
[[12 23 34]
[23 34 45]
[34 45 56]]
向量乘法运算：
[[ 11  42  93]
[ 42  93 164]
[ 93 164 255]]
```

NumPy 主要用来处理大量数字的应用，通过这三个例子，熟悉了 NumPy 的多维数组的定义、访问和向量运算。

4.3 Matplotlib 速成

Matplotlib 是 Python 中著名的 2D 绘图库，使用方法比较简单，按照下面的三步进行操作就能很简单地完成绘图。

- 调用 plot()、scatter()等方法，并为绘图填充数据。数据是 NumPy 的 ndarray 类型的对象。
- 设定数据标签，使用 xlabel()、ylabel()方法。
- 展示绘图结果，使用 show()方法。

4.3.1 绘制线条图

下面是一个简单的绘制线条图的例子，代码如下：

```
import matplotlib.pyplot as plt
import numpy as np
# 定义绘图的数据
myarray = np.array([[1, 2, 3], [2, 3, 4], [3, 4, 5]])
# 初始化绘图
plt.plot(myarray)
# 设定 x 轴和 y 轴
plt.xlabel('x axis')
plt.ylabel('y axis')
# 绘图
plt.show()
```

执行结果如图 4-1 所示。

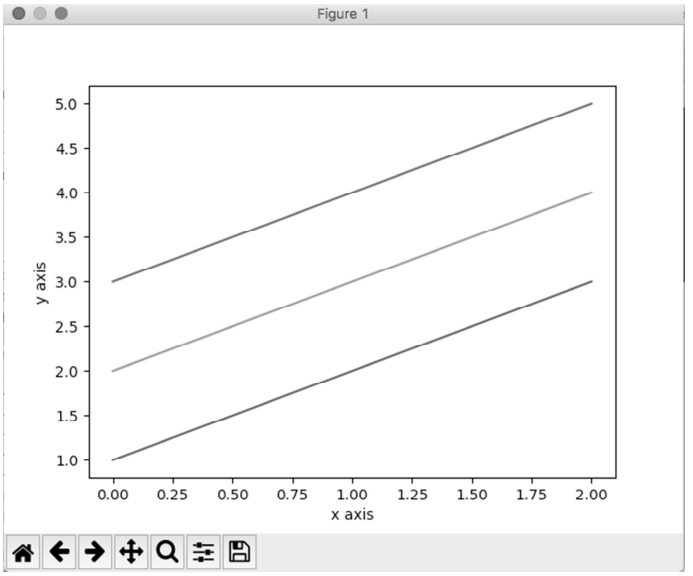


图 4-1

4.3.2 散点图

下面是一个简单的绘制散点图的例子，代码如下：

```
import matplotlib.pyplot as plt
import numpy as np
```

```
# 定义绘图的数据
myarray1 = np.array([1,2,3])
myarray2 = np.array([11,21,31])
# 初始化绘图
plt.scatter(myarray1, myarray2)
# 设定 x 轴和 y 轴
plt.xlabel('x axis')
plt.ylabel('y axis')
# 绘图
plt.show()
```

执行结果如图 4-2 所示。

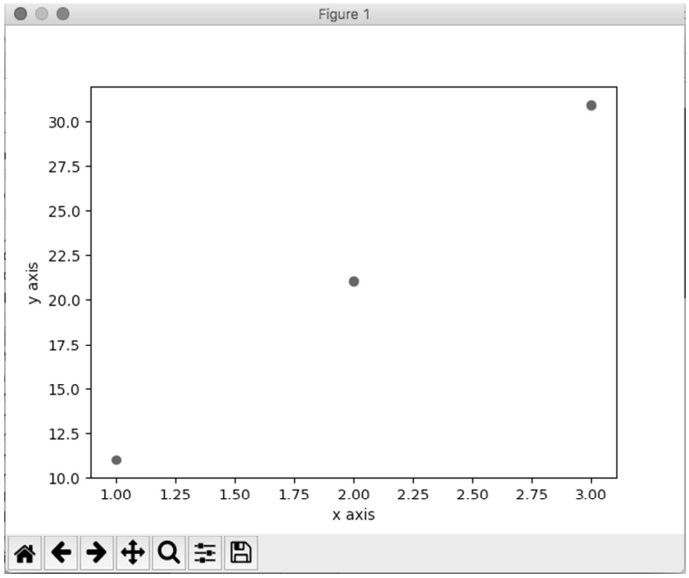


图 4-2

Matplotlib 提供了很多种类的图表的绘制功能。在 http://matplotlib.org/gallery.html 提供了超过 100 种示例，详细情况可以参考 Matplotlib API 的说明。

4.4 Pandas 速成

Pandas 提供了用于机器学习的复杂数据类型：矢量运算方法和数据分析方法。Pandas 也提供了多种数据结构。

- **Series：**一维数组，与 NumPy 中的一维 Array 类似。二者与 Python 基本的数据结构 List 也很相近，其区别是：List 中的元素可以是不同的数据类型，而 Array 和 Series 中则只允许存储相同的数据类型，这样可以更有效地使用内存，提高运算效率。
- **Time- Series：**以时间为索引的 Series。
- **DataFrame：**二维的表格型数据结构。很多功能与 R 语言中的 data.frame 类似。可以将 DataFrame 理解为 Series 的容器。
- **Panel：**三维数组，可以理解为 DataFrame 的容器。

在这里我们需要深入理解 Series 和 DataFrame 这两种数据类型，来帮助我们实践机器学习。

4.4.1 Series

Series 虽然与 NumPy 的一维数组类似，但是在建立 Series 时可以设定 index，也可以像访问 NumPy 数组或字典一样来访问 Series 元素。代码如下：

```
import numpy as np
import pandas as pd
myarray = np.array([1, 2, 3])
index = ['a', 'b', 'c']
myseries = pd.Series(myarray, index=index)
print(myseries)
print('Series 中的第一个元素：')
print(myseries[0])
print('Series 中的 c index 元素：')
print(myseries['c'])
```

执行结果如下：

```
a    1
b    2
c    3
dtype: int64
Series 中的第一个元素：
1
Series 中的 c index 元素：
3
```

4.4.2 DataFrame

DataFrame 是一个可以指定行和列标签的二维数组。数据可以通过指定列名来访问特定列的数据。代码如下：

```
import numpy as np
import pandas as pd
myarray = np.array([[1, 2, 3], [2, 3, 4], [3, 4, 5]])
rowindex = ['row1', 'row2', 'row3']
colname = ['col1', 'col2', 'col3']
mydataframe = pd.DataFrame(data=myarray, index=rowindex, columns=colname)
print(mydataframe)
print('访问 col3 的数据：')
print(mydataframe['col3'])
```

执行结果如下：

```
      col1  col2  col3
row1    1     2     3
row2    2     3     4
row3    3     4     5
访问 col3 的数据：
row1    3
row2    4
row3    5
Name: col3, dtype: int64
```

Pandas 是一个功能强大的对数据进行切片的工具，更多的内容请查阅 Pandas API（http://pandas.pydata.org/pandas-docs/stable/）。

4.5 总结

本章主要是熟悉 Python 的语法，以及在机器学习领域中使用的几个类库：SciPy、NumPy、Pandas 和 Matplotlib。下面将学习如何将数据加载到 Python 中，为机器学习项目准备数据。

第二部分

数据理解

“ 定量研究最重要的是如何提出问题，而不是数据和统计方法 ”“ 定量研究是一个全面的过程，不只是数据，也不只是统计，而是运用统计来研究数据，来回答我们想要了解的问题，数据、统计方法、理念，三者缺一不可，其中，理念是最重要的 ”“ 每一个研究问题涉及的领域很多，人不能画地为牢，而是要看研究的问题把你带到哪里；定量研究不是数据导向，不是方法导向，也不是学科导向，我比较提倡的是问题导向，提出一个问题，最终目的是回答这个问题 ”。

—— 谢宇

“ 统计学是什么？概率与数学。用概率与数学来分析人，分析的永远不是人。用永远不是人的结论指导人实在是一种偏误。在这个意义上讲，解读强于技术 ”“ 理解人比什么都重要，方法、技术、手段、学科都仅仅是背景 ”“ 定量研究是分析趋势，是找寻和社会相适应的一个概括视角，这才是定量研究。如果你想把数据做好，先要理解社会、理解人、理解营销，才能谈得上定量 ”。

——刘德寰

5 数据导入

在训练机器学习的模型时，需要用到大量数据，最常用的做法是利用历史的数据来训练模型。这些数据通常会以 CSV 的格式来存储，或者能够方便地转化为 CSV 格式。在开始启动机器学习项目之前，必须先将数据导入到 Python 中。下面将介绍三种将 CSV 数据导入到 Python 中的方法，以便完成对机器学习算法的训练。

- 通过标准的 Python 库导入 CSV 文件。
- 通过 NumPy 导入 CSV 文件。
- 通过 Pandas 导入 CSV 文件。

5.1 CSV 文件

CSV 文件是用逗号（,）分隔的文本文件。在数据导入之前，通常会审查一下 CSV 文件中包含的内容。在审查 CSV 文件时，通常要注意以下几个方面。

5.1.1 文件头

如果 CSV 的文件里包括文件头的信息，可以很方便地使用文件头信息来设置读入数据字段的属性名称。如果文件里不含有文件头信息，需要自己手动设定读入文件的字段属性名称。数据导入时，设置字段属性名称，有助于提高数据处理程序的可读性。

5.1.2 文件中的注释

在 CSV 文件中，注释行是以“井”号（#）开头的。是否需要对读入的注释行做处理，取决于采用什么方式读入 CSV 文件。

5.1.3 分隔符

CSV 文件的标准分隔符是逗号（,），当然也可以使用 Tab 键或空格键作为自定义的分隔符。当使用这两种分隔符时，文件读取是要指明分隔符的。

5.1.4 引号

当有的字段值中有空白时，这些值通常都会被引号引起来，默认使用双引号来标记这些字段值。如果采用自定义格式，那么在文件读取时要明确在文件中采用的自定义格式。

5.2 Pima Indians 数据集

首先介绍一下在本章和后续章节中要使用的测试数据。目前在 UCI 机器学习仓库（http://archive.ics.uci.edu/ml/datasets.html）中有大量的免费数据，可以利用这些数据来学习机器学习，并训练算法模型 。本章选择的 Pima Indians 数据集就是从 UCI 中获取的。这是一个分类问题的数据集，主要记录了印第安人最近五年内是否患糖尿病的医疗数据。这些数据都是以数字的方式记录的，并且输出结果是 0 或 1，使我们在机器学习的算法中建立模型变得非常方便。

5.3 采用标准 Python 类库导入数据

Python 提供了一个标准类库 CSV，用来处理 CSV 文件。这个类库中的 reader()函数用来读入 CSV 文件。当 CSV 文件被读入后，可以利用这些数据生成一个 NumPy 数组，用来训练算法模型。首先下载数据文件到应用目录下，并命名文件为 pima.csv。这个文件中所有的数据都是数字，并且数据中不含有文件头。代码如下：

```
from csv import reader
import numpy as np
# 使用标准的 Python 类库导入 CSV 数据
filename = 'pima_data.csv'
with open(filename, 'rt') as raw_data:
    readers = reader(raw_data, delimiter=',')
    x = list(readers)
    data = np.array(x).astype('float')
    print(data.shape)
```

执行结果如下：

```
(768, 9)
```

代码非常简单，就不详细进行讲解了。详细内容请查阅 Python 的 API 介绍。

5.4 采用 NumPy 导入数据

也可以使用 NumPy 的 loadtxt()函数导入数据。使用这个函数处理的数据没有文件头，并且所有的数据结构是一样的，也就是说，数据类型是一样的。代码如下：

```
from numpy import loadtxt
# 使用 NumPy 导入 CSV 数据
filename = 'pima_data.csv'
with open(filename, 'rt') as raw_data:
    data = loadtxt(raw_data, delimiter=',')
    print(data.shape)
```

这段代码就比直接使用 Python 的标准类库简洁了很多，执行结果如下：

```
(768, 9)
```

5.5 采用 Pandas 导入数据

前面介绍了如何通过标准的 Python 类库和 NumPy 来导入数据。接下来会通过一个例子来演示如何通过 Pandas 导入 CSV 文件的数据。通过 Pandas 来导入 CSV 文件要使用 pandas.read_csv()函数。这个函数的返回值是 DataFrame，可以很方便地进行下一步的处理。这个函数的名称非常直观，便于代码的阅读和后续对数据的处理。在机器学习的项目中，经常利用 Pandas 来做数据清洗与数据准备工作。因此，在导入 CSV 文件时，推荐大家使用这个方法。代码如下：

```
from pandas import read_csv
# 使用 Pandas 导入 CSV 数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data = read_csv(filename, names=names)
print(data.shape)
```

这段代码为数据补充了文件头，执行结果如下：

```
(768, 9)
```

5.6 总结

本章主要介绍了三种导入 CSV 文件到 Python 的方法，分别是：通过标准的 Python 类库导入、通过 NumPy 导入和通过 Pandas 导入。在进行机器学习项目实践时，建议采用 Pandas 方式导入数据。到目前为止，本书已经介绍了在机器学习中的基本环境和相应的类库的使用方法，以及一个简单的机器学习的分类的例子。本章介绍了如何导入数据到 Python 中，接下来就通过描述统计的方式来理解导入的数据。

6 数据理解

为了得到更准确的结果，必须理解数据的特征、分布情况，以及需要解决的问题，以便建立和优化算法模型。本章将介绍七种方法来帮助大家理解数据。

- 简单地查看数据。
- 审查数据的维度。
- 审查数据的类型和属性。
- 总结查看数据分类的分布情况。
- 通过描述性统计分析数据。
- 理解数据属性的相关性。
- 审查数据的分布状况。

6.1 简单地查看数据

对数据的简单审视，是加强对数据理解最有效的方法之一。通过对数据的观察，可以发现数据的内在关系。这些发现有助于对数据进行整理。接下来通过一个简单的例子展示一下如何查看数据。这个例子是查看前 10 行数据。代码如下：

```
from pandas import read_csv
# 显示数据的前 10 行
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data = read_csv(filename, names=names)
peek = data.head(10)
print(peek)
```

执行结果如图 6-1 所示。

	preg	plas	pres	skin	test	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1

图 6-1

6.2 数据的维度

在机器学习中要注意数据的行和列，必须对所拥有的数据非常了解，要知道有多少行和多少列，这是因为：

- 太多的行会导致花费大量时间来训练算法得到模型；太少的数据会导致对算法的训练不充分，得不到合适的模型。
- 如果数据具有太多的特征，会引起某些算法性能低下的问题。

通过 DataFrame 的 shape 属性，可以很方便地查看数据集中有多少行和多少列。代码如下：

```
from pandas import read_csv
# 显示数据的行和列数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
```

```
data = read_csv(filename, names=names)
print(data.shape)
```

执行结果如下：

```
(768, 9)
```

6.3 数据属性和类型

数据的类型是很重要的一个属性。字符串会被转化成浮点数或整数，以便于计算和分类。可以通过 DataFrame 的 Type 属性来查看每一个字段的数据类型。代码如下：

```
from pandas import read_csv
# 显示数据的行和列数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data = read_csv(filename, names=names)
print(data.dtypes)
```

执行结果如下：

```
preg     int64
plas     int64
pres     int64
skin     int64
test     int64
mass   float64
pedi   float64
age      int64
class    int64
dtype: object
```

6.4 描述性统计

描述性统计可以给出一个更加直观、更加清晰的视角，以加强对数据的理解。在这里可以通过 DataFrame 的 describe()方法来查看描述性统计的内容。这个方法给我们展示

了八方面的信息：数据记录数、平均值、标准方差、最小值、下四分位数、中位数、上四分位数、最大值。这些信息主要用来描述数据的分布情况。代码如下：

```
from pandas import read_csv
from pandas import set_option
# 描述性统计
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data = read_csv(filename, names=names)
set_option('display.width', 100)
# 设置数据的精确度
set_option('precision', 4)
print(data.describe())
```

执行结果如图 6-2 所示。

	preg	plas	pres	skin	test	mass	pedi	age	class
count	768.0000	768.0000	768.0000	768.0000	768.0000	768.0000	768.0000	768.0000	768.000
mean	3.8451	120.8945	69.1055	20.5365	79.7995	31.9926	0.4719	33.2409	0.349
std	3.3696	31.9726	19.3558	15.9522	115.2440	7.8842	0.3313	11.7602	0.477
min	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0780	21.0000	0.000
25%	1.0000	99.0000	62.0000	0.0000	0.0000	27.3000	0.2437	24.0000	0.000
50%	3.0000	117.0000	72.0000	23.0000	30.5000	32.0000	0.3725	29.0000	0.000
75%	6.0000	140.2500	80.0000	32.0000	127.2500	36.6000	0.6262	41.0000	1.000
max	17.0000	199.0000	122.0000	99.0000	846.0000	67.1000	2.4200	81.0000	1.000

图 6-2

6.5 数据分组分布（适用于分类算法）

在分类算法中，需要知道每个分类的数据大概有多少条记录，以及数据分布是否平衡。如果数据分布的平衡性很差，需要在数据加工阶段进行数据处理，来提高数据分布的平衡性。利用 Pandas 的属性和方法，可以很方便地查看数据的分布情况。代码如下：

```
from pandas import read_csv
# 数据分类分布统计
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
```

```
data = read_csv(filename, names=names)
print(data.groupby('class').size())
```

执行结果如下：

```
class
0   500
1   268
dtype: int64
```

6.6 数据属性的相关性

数据属性的相关性是指数据的两个属性是否互相影响，以及这种影响是什么方式的等。非常通用的计算两个属性的相关性的方法是皮尔逊相关系数，皮尔逊相关系数是度量两个变量间相关程度的方法。它是一个介于 1 和 -1 之间的值，其中，1 表示变量完全正相关， 0 表示无关，-1 表示完全负相关。在自然科学领域中，该系数广泛用于度量两个变量之间的相关程度。在机器学习中，当数据的关联性比较高时，有些算法（如linear、逻辑回归算法等）的性能会降低。所以在开始训练算法前，查看一下算法的关联性是一个很好的方法。当数据特征的相关性比较高时，应该考虑对特征进行降维处理。下面通过使用 DataFrame 的 corr()方法来计算数据集中数据属性之间的关联关系矩阵。代码如下：

```
from pandas import read_csv
from pandas import set_option
# 显示数据的相关性
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data = read_csv(filename, names=names)
set_option('display.width', 100)
# 设置数据的精确度
set_option('precision', 2)
print(data.corr(method='pearson'))
```

执行之后会得到一个每个属性相互影响的矩阵。执行结果如图 6-3 所示。

	preg	plas	pres	skin	test	mass	pedi	age	class
preg	1.00	0.13	0.14	-0.08	-0.07	0.02	-0.03	0.54	0.22
plas	0.13	1.00	0.15	0.06	0.33	0.22	0.14	0.26	0.47
pres	0.14	0.15	1.00	0.21	0.09	0.28	0.04	0.24	0.07
skin	-0.08	0.06	0.21	1.00	0.44	0.39	0.18	-0.11	0.07
test	-0.07	0.33	0.09	0.44	1.00	0.20	0.19	-0.04	0.13
mass	0.02	0.22	0.28	0.39	0.20	1.00	0.14	0.04	0.29
pedi	-0.03	0.14	0.04	0.18	0.19	0.14	1.00	0.03	0.17
age	0.54	0.26	0.24	-0.11	-0.04	0.04	0.03	1.00	0.24
class	0.22	0.47	0.07	0.07	0.13	0.29	0.17	0.24	1.00

图 6-3

6.7 数据的分布分析

通过分析数据的高斯分布情况来确认数据的偏离情况。高斯分布又叫正态分布，是在数据、物理及工程等领域都非常重要的概率分布，在统计学的许多方面有着重大的影响。高斯分布的曲线呈钟形，两头低，中间高，左右对称。在高斯分布图中，*y* 轴两点之间的面积是发生的概率。在很多机器学习算法中都会假定数据遵循高斯分布，先计算数据的高斯偏离状况，再根据偏离状况准备数据。我们可以使用 DataFrame 的 skew()方法来计算所有数据属性的高斯分布偏离情况。代码如下：

```
from pandas import read_csv
# 计算数据的高斯偏离
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data = read_csv(filename, names=names)
print(data.skew())
```

skew()函数的结果显示了数据分布是左偏还是右偏。当数据接近 0 时，表示数据的偏差非常小。执行结果如下：

```
preg    0.901674
plas    0.173754
pres    -1.843608
skin    0.109372
test    2.272251
mass    -0.428982
pedi    1.919911
```

```
age     1.129597
class    0.635017
dtype: float64
```

6.8 总结

本章学习了如何分析数据和理解数据。在审查数据的时候，需要记住以下几个小技巧。

- **审查数字**：通常描述性分析给出的数据对数据的理解是不充分的，应该观察思考数据的特点，找到数据的内在联系和对解决问题有什么益处。
- **问为什么**：审查数据后多问几个“为什么”，如你是如何看到和为什么看到这些数字的特殊性的，这些数字和问题如何关联到一起的，以及这些数字和我们的问题有什么关系等。
- **写下想法**：写下自己通过观察得到的想法，用便笺、记事本等将观察到的数据各维度的关联关系和我们的想法都记录下来。例如，数字代表什么，将采用什么样的技术继续挖掘数据等。写下的这些想法将会给新的尝试带来极大的参考价值。

对数据的分析是机器学习中的重要领域，通过对数据的理解，可以选择有效的算法来建立模型。本章介绍了七种方法来观察和理解数据，这为发现数据的特征和选择合适的算法提供了思路。在接下来的一章中，会介绍另外一种观察数据的有效手段，就是通过 Matplotlib 提供的可视化图表来展示数据，以便于发现数据的特征。

7 数据可视化

为了生成最优化的算法模型，必须对数据进行理解。最快、最有效的方式是通过数据的可视化来加强对数据的理解。在这一章将介绍如何通过 Matplotlib 来可视化数据，以加强对数据的理解。本章使用的数据集是在第 5 章中使用过的 Pima Indians 数据集。

7.1 单一图表

本章将通过以下三种图表来展示数据：

- 直方图。
- 密度图。
- 箱线图。

7.1.1 直方图

直方图（Histogram）又称质量分布图，是一种统计报告图，由一系列高度不等的纵向条纹或线段表示数据的分布情况。一般用横轴表示数据类型，纵轴表示分布情况。直

方图可以非常直观地展示每个属性的分布状况。通过图表可以很直观地看到数据是高斯分布、指数分布还是偏态分布。代码如下：

```
from pandas import read_csv
import matplotlib.pyplot as plt
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data = read_csv(filename, names=names)
data.hist()
plt.show()
```

执行结果如图 7-1 所示，我们可以看到，age、pedi 和 test 也许是指数分布；mass、pres 和 plas 也许是高斯分布。

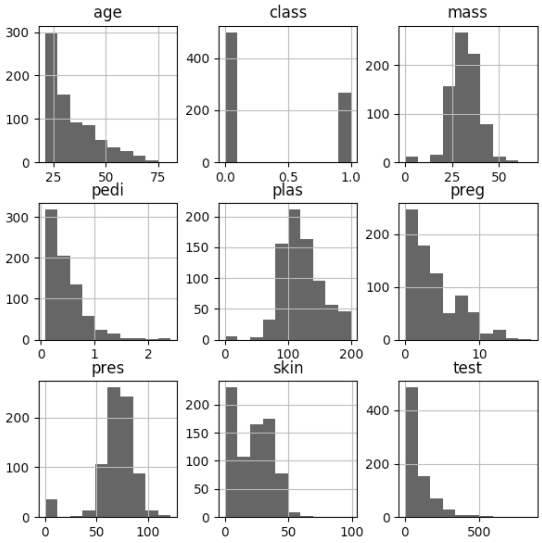


图 7-1

7.1.2 密度图

密度图是一种表现与数据值对应的边界或域对象的图形表示方法，一般用于呈现连续变量。密度图类似于对直方图进行抽象，用平滑的线来描述数据的分布。这也是一种用来显示数据分布的图表。代码如下：

```
from pandas import read_csv
import matplotlib.pyplot as plt
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data = read_csv(filename, names=names)
data.plot(kind='density', subplots=True, layout=(3,3), sharex=False)
plt.show()
```

通过密度图来显示数据的分布，相对于直方图更直观。执行结果如图 7-2 所示。

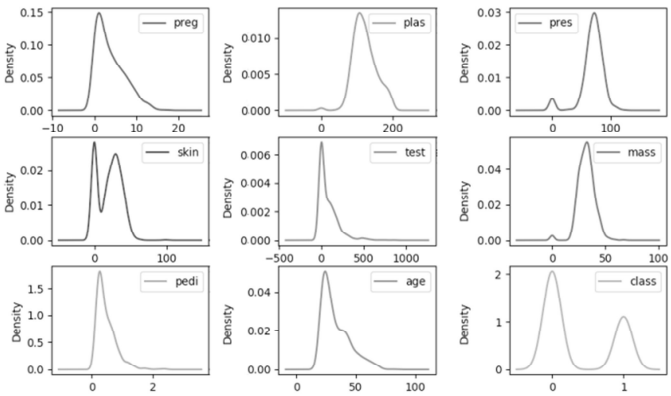


图 7-2

7.1.3 箱线图

箱线图又称盒须图、盒式图或箱形图，是一种用于显示一组数据分散情况的统计图。因形状如箱子而得名，在各种领域都经常被使用。箱线图也是一种非常好的用于显示数据分布状况的手段。首先画一条中位数线，然后以下四分位数和上四分位数画一个盒子，上下各有一条横线，表示上边缘和下边缘，通过横线来显示数据的伸展状况，游离在边缘之外的点为异常值。代码如下：

```
from pandas import read_csv
import matplotlib.pyplot as plt
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
```

```
'class']
data = read_csv(filename, names=names)
data.plot(kind='box', subplots=True, layout=(3,3), sharex=False)
plt.show()
```

通过图表可以看到不同属性的延伸截然不同。执行结果如图 7-3 所示。

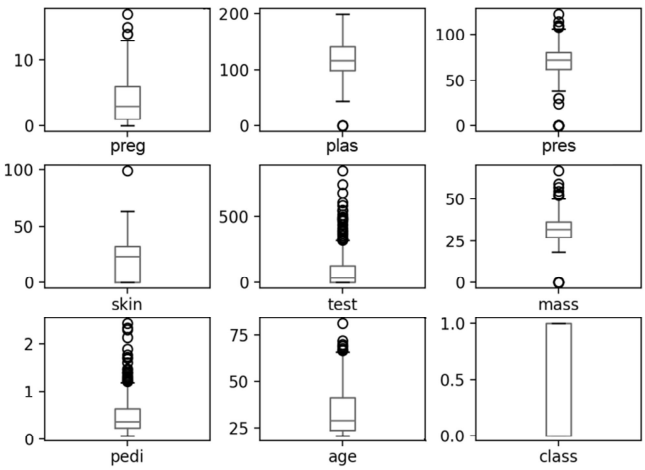


图 7-3

7.2 多重图表

接下来将介绍两种图表，以显示不同属性之间的关联关系：相关矩阵图和散点矩阵图。

7.2.1 相关矩阵图

相关矩阵图主要用来展示两个不同属性相互影响的程度。如果两个属性按照相同的方向变化，说明是正向影响。如果两个属性朝相反方向变化，说明是反向影响。把所有属性两两影响的关系展示出来的图表就叫相关矩阵图。矩阵图法就是从多维问题的事件中找出成对的因素，排列成矩阵图，然后根据矩阵图来分析问题，确定关键点。它是一种通过多因素综合思考来探索问题的好方法。代码如下：

```
from pandas import read_csv
import matplotlib.pyplot as plt
import numpy as np
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data = read_csv(filename, names=names)
correlations = data.corr()
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = np.arange(0, 9, 1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(names)
ax.set_yticklabels(names)
plt.show()
```

在图表的左边和上边显示的是完全相同的属性名称，通过这个矩阵可以很清楚地看到各个属性两两关联的关系。执行结果如图 7-4 所示。

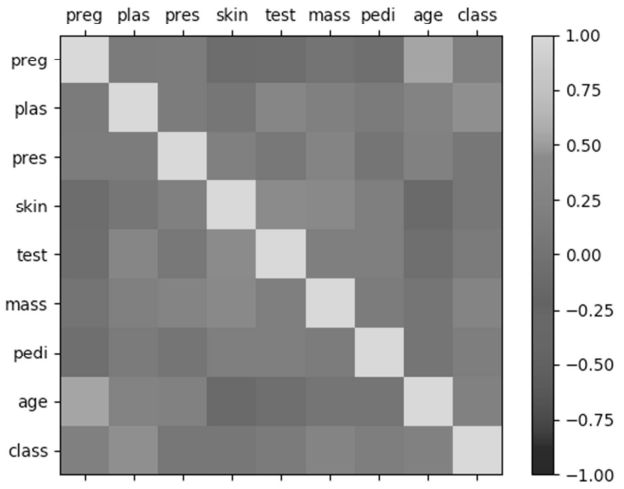


图 7-4

7.2.2 散点矩阵图

散点矩阵图表示因变量随自变量变化的大致趋势，据此可以选择合适的函数对数据点进行拟合。散点矩阵图由两组数据构成多个坐标点，考察坐标点的分布，可以判断两个变量之间是否存在某种关联或总结坐标点的分布模式。散点矩阵图将序列显示为一组点，值由点在图表中的位置表示，类别由图表中的不同标记表示。散点矩阵图通常用于比较跨类别的聚合数据。当同时考察多个变量的相关关系时，若一一绘制它们的简单散点图将十分麻烦。此时可利用散点矩阵图来绘制各个变量间的散点图，这样可以快速发现多个变量间的主要相关性，这在进行多元线性回归时显得尤为重要。代码如下：

```
from pandas import read_csv
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data = read_csv(filename, names=names)
scatter_matrix(data)
plt.show()
```

执行结果如图 7-5 所示。

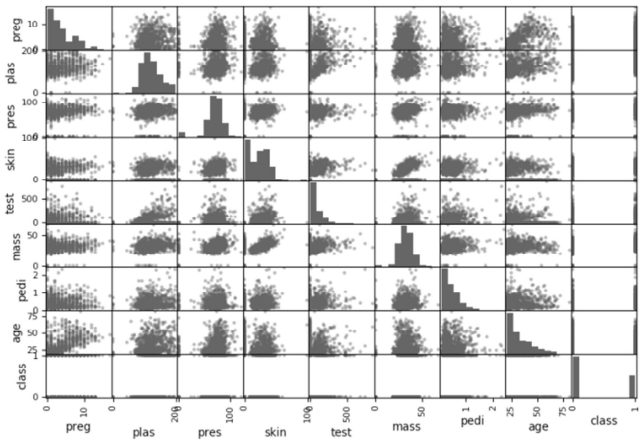


图 7-5

7.3 总结

本章介绍了几种图表，用于展示数据的分布状况和两两之间的影响关系。结合前一章介绍的方法，我们对数据的理解更加深入了，解决问题的思路也更加清晰了。下一章将会介绍利用这些分析得出的结论进行数据整理。

第三部分

数据准备

特征选择是困难耗时的，也需要对需求的理解和专业知识的掌握。在机器学习的应用开发中，最基础的是特征工程。

——吴恩达

吴恩达老师的这句话充分概括了特征工程的复杂度及其重要性。Kagglers 比赛和天池比赛的冠军其实在比赛中并没有用到很高深的算法，大多数都是在特征工程这个环节做了出色的工作，然后使用一些常见的算法，如逻辑回归，就能得到性能出色的模型。因此特征工程是建立高准确度机器学习算法的基础。所以也有机器学习方面的专家这样来概括机器学习：“使用正确的特征来构建正确的模型，以完成既定的任务”。

8 数据预处理

数据预处理需要根据数据本身的特性进行，有不同的格式和不同的要求，有缺失值的要填，有无效数据的要剔，有冗余维的要选，这些步骤都和数据本身的特性紧密相关。数据预处理大致分为三个步骤：数据的准备、数据的转换、数据的输出。数据处理是系统工程的基本环节，也是提高算法准确度的有效手段。因此，为了提高算法模型的准确度，在机器学习中也要根据算法的特征和数据的特征对数据进行转换。本章将利用 scikit-learn 来转换数据，以便我们将处理后的数据应用到算法中，这样也可以提高算法模型的准确度。本章将介绍以下几种数据转换方法：

- 调整数据尺度（Rescale Data）。
- 正态化数据（Standardize Data）。
- 标准化数据（Normalize Data）。
- 二值数据（Binarize Data）。

8.1 为什么需要数据预处理

在开始机器学习的模型训练之前，需要对数据进行预处理，这是一个必需的过

程。但是需要注意的是，不同的算法对数据有不同的假定，需要按照不同的方式转换数据。当然，如果按照算法的规则来准备数据，算法就可以产生一个准确度比较高的模型。

8.2 格式化数据

本章会介绍四种不同的方法来格式化数据依然使用 Pima Indian 的数据集作为例子。这四种方法都会按照统一的流程来处理数据：

- 导入数据。
- 按照算法的输入和输出整理数据。
- 格式化输入数据。
- 总结显示数据的变化。

scikit-learn 提供了两种标准的格式化数据的方法，每一种方法都有适用的算法。利用这两种方法整理的数据，可以直接用来训练算法模型。在 scikit-learn 的说明文档中，也有对这两种方法的详细说明：

- 适合和多重变换（Fit and Multiple Transform）。
- 适合和变换组合（Combined Fit-and-Transform）。

推荐优先选择适合和多重变换（Fit and Multiple Transform）方法。首先调用 fit()函数来准备数据转换的参数，然后调用 transform()函数来做数据的预处理。适合和变换组合（Combined Fit-and-Transform）对绘图或汇总处理具有非常好的效果。详细的数据预处理方法可以参考 scikit-learn 的 API 文档。

8.3 调整数据尺度

如果数据的各个属性按照不同的方式度量数据，那么通过调整数据的尺度让所有的属性按照相同的尺度来度量数据，就会给机器学习的算法模型训练带来极大的方便。这个方法通常会将数据的所有属性标准化，并将数据转换成 0 和 1 之间的值，这对于梯度

下降等算法是非常有用的，对于回归算法、神经网络算法和 K 近邻算法的准确度提高也起到很重要的作用。

在统计学中，按照对事物描述的精确度，对所采用的尺度从低级到高级分成四个层次：定类尺度、定序尺度、定距尺度和定比尺度。定类尺度是对事物类别属性的一种测度，按照事物的属性进行分组或分类。定序尺度是对事物之间的等级或顺序的一种测度，可以比较优劣或排序。定距尺度和定比尺度是对事物类别或次序之间间距的测量，定距尺度的特点是其不仅能将事物区分为不同的类型并进行排序，而且可以准确地指出类别之间的差距。而定比尺度则更近一步，它和定距尺度的差别在于它有一个固定的绝对“零”点。由于这两种测量尺度在绝大多数统计分析中没有本质的差别，所以很多时候都没有严格的区分。

在 scikit-learn 中，可以通过 MinMaxScaler 类来调整数据尺度。将不同计量单位的数据统一成相同的尺度，利于对事物的分类或分组。实际上，MinMaxScaler 是将属性缩放到一个指定范围，或者对数据进行标准化并将数据都聚集到 0 附近，方差为 1。数据尺度的统一，通常能够提高与距离相关的算法的准确度（如 K 近邻算法）。下面给出一个对数据进行缩放的例子。代码如下：

```
 # 调整数据尺度（0..）
from pandas import read_csv
from numpy import set_printoptions
from sklearn.preprocessing import MinMaxScaler
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
transformer = MinMaxScaler(feature_range=(0, 1))
# 数据转换
newX = transformer.fit_transform(X)
# 设定数据的打印格式
```

```
set_printoptions(precision=3)
print(newX)
```

调整完数据的尺度之后，所有的数据都按照设定的分布区间进行分布。执行结果如下：

```
[[ 0.353  0.744  0.59  ...,  0.501  0.234  0.483]
 [ 0.059  0.427  0.541 ...,  0.396  0.117  0.167]
 [ 0.471  0.92   0.525 ...,  0.347  0.254  0.183]
 ...,
 [ 0.294  0.608  0.59  ...,  0.39   0.071  0.15 ]
 [ 0.059  0.633  0.492 ...,  0.449  0.116  0.433]
 [ 0.059  0.467  0.574 ...,  0.453  0.101  0.033]]
```

8.4 正态化数据

正态化数据（Standardize Data）是有效的处理符合高斯分布的数据的手段，输出结果以 0 为中位数，方差为 1，并作为假定数据符合高斯分布的算法的输入。这些算法有线性回归、逻辑回归和线性判别分析等。在这里可以通过 scikit-learn 提供的 StandardScaler 类来进行正态化数据处理。代码如下：

```
 # 正态化数据
from pandas import read_csv
from numpy import set_printoptions
from sklearn.preprocessing import StandardScaler
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
transformer = StandardScaler().fit(X)
# 数据转换
newX = transformer.transform(X)
# 设定数据的打印格式
```

```
set_printoptions(precision=3)
print(newX)
```

输出结果如下：

```
[[ 0.64   0.848  0.15  ...,  0.204  0.468  1.426]
 [-0.845 -1.123 -0.161 ..., -0.684 -0.365 -0.191]
 [ 1.234  1.944 -0.264 ..., -1.103  0.604 -0.106]
 ...,
 [ 0.343  0.003  0.15  ..., -0.735 -0.685 -0.276]
 [-0.845  0.16  -0.471 ..., -0.24  -0.371  1.171]
 [-0.845 -0.873  0.046 ..., -0.202 -0.474 -0.871]]
```

8.5 标准化数据

标准化数据（Normalize Data）处理是将每一行的数据的距离处理成 1（在线性代数中矢量距离为 1）的数据又叫作“归一元”处理，适合处理稀疏数据（具有很多为 0 的数据），归一元处理的数据对使用权重输入的神经网络和使用距离的 K 近邻算法的准确度的提升有显著作用。使用 scikit-learn 中的 Normalizer 类实现。代码如下：

```
 # 标准化数据
from pandas import read_csv
from numpy import set_printoptions
from sklearn.preprocessing import Normalizer
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
transformer = Normalizer().fit(X)
# 数据转换
newX = transformer.transform(X)
# 设定数据的打印格式
```

```
set_printoptions(precision=3)
print(newX)
```

执行结果如下：

```
[[ 0.034  0.828  0.403 ...,  0.188  0.004  0.28 ]
 [ 0.008  0.716  0.556 ...,  0.224  0.003  0.261]
 [ 0.04   0.924  0.323 ...,  0.118  0.003  0.162]
 ...,
 [ 0.027  0.651  0.388 ...,  0.141  0.001  0.161]
 [ 0.007  0.838  0.399 ...,  0.2    0.002  0.313]
 [ 0.008  0.736  0.554 ...,  0.241  0.002  0.182]]
```

8.6 二值数据

二值数据（Binarize Data）是使用值将数据转化为二值，大于阈值设置为 1，小于阈值设置为 0。这个过程被叫作二分数据或阈值转换。在生成明确值或特征工程增加属性的时候使用，使用 scikit-learn 中的 Binarizer 类实现。代码如下：

```
# 二值数据
from pandas import read_csv
from numpy import set_printoptions
from sklearn.preprocessing import Binarizer
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
transformer = Binarizer(threshold=0.0).fit(X)
# 数据转换
newX = transformer.transform(X)
# 设定数据的打印格式
set_printoptions(precision=3)
print(newX)
```

执行结果如下：

```
[[ 1.  1.  1. ...,  1.  1.  1.]
 [ 1.  1.  1. ...,  1.  1.  1.]
 [ 1.  1.  1. ...,  1.  1.  1.]
 ...,
 [ 1.  1.  1. ...,  1.  1.  1.]
 [ 1.  1.  1. ...,  1.  1.  1.]
 [ 1.  1.  1. ...,  1.  1.  1.]]
```

8.7 总结

在这一章学到了在 scikit-learn 中对数据进行预处理的四种方法。这四种方法适用于不同的场景，可以在实践中根据不同的算法模型来选择不同的数据预处理方法。下一章将会学习如何选择数据的特征属性来训练预测（分类与回归）模型。

9 数据特征选定

在做数据挖掘和数据分析时，数据是所有问题的基础，并且会影响整个项目的进程。相较于使用一些复杂的算法，灵活地处理数据经常会取到意想不到的效果。而处理数据不可避免地会使用到特征工程。那么特征工程是什么呢？有这么一句话在业界广为流传：数据和特征决定了机器学习的上限，而模型和算法只是逼近这个上限而已。因此，特征过程的本质就是一项工程活动，目的是最大限度地从原始数据中提取合适的特征，以供算法和模型使用。特征处理是特征工程的核心部分，scikit-learn 提供了较为完整的特征处理方法，包括数据预处理、特征选择、降维等。

本章将会学习通过 scikit-learn 来自动选择用于建立机器学习模型的数据特征的方法。本章将会介绍以下四个数据特征选择的方法：

- 单变量特征选定。
- 递归特征消除。
- 主要成分分析。
- 特征的重要性。

9.1 特征选定

特征选定是一个流程，能够选择有助于提高预测结果准确度的特征数据，或者有助于发现我们感兴趣的输出结果的特征数据。如果数据中包含无关的特征属性，会降低算法的准确度，对预测新数据造成干扰，尤其是线性相关算法（如线性回归算法和逻辑回归算法）。因此，在开始建立模型之前，执行特征选定有助于：

降低数据的拟合度：较少的冗余数据，会使算法得出结论的机会更大。

提高算法精度：较少的误导数据，能够提高算法的准确度。

减少训练时间：越少的数据，训练模型所需要的时间越少。

可以在 scikit-learn 的特征选定文档中查看更多的信息（http://scikit-learn.org/stable/modules/ feature_selection.html）。下面我们会继续使用 Pima Indians 的数据集来进行演示。

9.2 单变量特征选定

统计分析可以用来分析选择对结果影响最大的数据特征。在 scikit-learn 中提供了 SelectKBest 类，可以使用一系列统计方法来选定数据特征，是对卡方检验的实现。经典的卡方检验是检验定性自变量对定性因变量的相关性的方法。假设自变量有 N 种取值，因变量有 M 种取值，考虑自变量等于 i 且因变量等于 j 的样本频数的观察值与期望值的差距，构建统计量。卡方检验就是统计样本的实际观测值与理论推断值之间的偏离程度，偏离程度决定了卡方值的大小，卡方值越大，越不符合；卡方值越小，偏差越小，越趋于符合；若两个值完全相等，卡方值就为 0，表明理论值完全符合。下面的例子是通过卡方检验（chi-squared）的方式来选择四个对结果影响最大的数据特征。代码如下：

```
 # 通过卡方检验选定数据特征
from pandas import read_csv
from numpy import set_printoptions
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
# 导入数据
```

```
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
# 特征选定
test = SelectKBest(score_func=chi2, k=4)
fit = test.fit(X, Y)
set_printoptions(precision=3)
print(fit.scores_)
features = fit.transform(X)
print(features)
```

执行结束后，我们得到了卡方检验对每一个数据特征的评分，以及得分最高的四个数据特征。执行结果如下：

```
[  111.52   1411.887    17.605    53.108  2175.565   127.669     5.393
   181.304]
[[ 148.     0.    33.6   50. ]
 [  85.     0.    26.6   31. ]
 [ 183.     0.    23.3   32. ]
 ...,
 [ 121.   112.    26.2   30. ]
 [ 126.     0.    30.1   47. ]
 [  93.     0.    30.4   23. ]]
```

通过设置 SelectKBest 的 score_func 参数，SelectKBest 不仅可以执行卡方检验来选择数据特征，还可以通过相关系数、互信息法等统计方法来选定数据特征。具体情况请参阅 scikit-learn 的说明文档。

9.3 递归特征消除

递归特征消除（RFE）使用一个基模型来进行多轮训练，每轮训练后消除若干权值系数的特征，再基于新的特征集进行下一轮训练。通过每一个基模型的精度，找到对最

终的预测结果影响最大的数据特征。在 scikit-learn 文档中有更多的关于递归特征消除（RFE）的描述。下面的例子是以逻辑回归算法为基模型，通过递归特征消除来选定对预测结果影响最大的三个数据特征。代码如下：

```
 # 通过递归消除来选定特征
from pandas import read_csv
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
# 特征选定
model = LogisticRegression()
rfe = RFE(model, 3)
fit = rfe.fit(X, Y)
print("特征个数：")
print(fit.n_features_)
print("被选定的特征：")
print(fit.support_)
print("特征排名：")
print(fit.ranking_)
```

执行后，我们可以看到 RFE 选定了 preg、mass 和 pedi 三个数据特征，它们在 support_ 中被标记为 True，在 ranking_中被标记为 1。执行结果如下：

```
特征个数：
3
被选定的特征：
[ True False False False False  True  True False]
特征排名：
[1 2 3 5 6 1 1 4]
```

9.4 主要成分分析

主要成分分析（PCA）是使用线性代数来转换压缩数据，通常被称作数据降维。 常见的降维方法除了主要成分分析（PCA），还有线性判别分析（LDA），它本身也是一个分类模型。PCA 和 LDA 有很多的相似之处，其本质是将原始的样本映射到维度更低的样本空间中，但是 PCA 和 LDA 的映射目标不一样：PCA 是为了让映射后的样本具有最大的发散性；而 LDA 是为了让映射后的样本有最好的分类性能。所以说，PCA 是一种无监督的降维方法，而 LDA 是一种有监督的降维方法。在聚类算法中，通常会利用 PCA 对数据进行降维处理，以利于对数据的简化分析和可视化。详细内容请参考 scikit-learn 的 API 文档。代码如下：

```
# 通过主要成分分析选定数据特征
from pandas import read_csv
from sklearn.decomposition import PCA
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
# 特征选定
pca = PCA(n_components=3)
fit = pca.fit(X)
print("解释方差：%s" % fit.explained_variance_ratio_)
print(fit.components_)
```

执行结果如下：

```
解释方差：[ 0.88854663  0.06159078  0.02579012]
[[ -2.02176587e-03   9.78115765e-02   1.60930503e-02   6.07566861e-02
    9.93110844e-01   1.40108085e-02   5.37167919e-04  -3.56474430e-03]
 [ -2.26488861e-02  -9.72210040e-01  -1.41909330e-01   5.78614699e-02
    9.46266913e-02  -4.69729766e-02  -8.16804621e-04  -1.40168181e-01]
 [ -2.24649003e-02   1.43428710e-01  -9.22467192e-01  -3.07013055e-01
    2.09773019e-02  -1.32444542e-01  -6.39983017e-04  -1.25454310e-01]]
```

9.5 特征重要性

袋装决策树算法（Bagged Decision Tress）、随机森林算法和极端随机树算法都可以用来计算数据特征的重要性。这三个算法都是集成算法中的袋装算法，在后面的集成算法章节会有详细的介绍。下面给出一个使用 ExtraTreesClassifier 类进行特征的重要性计算的例子。代码如下：

```
# 通过决策树计算特征的重要性
from pandas import read_csv
from sklearn.ensemble import ExtraTreesClassifier
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
# 特征选定
model = ExtraTreesClassifier()
fit = model.fit(X, Y)
print(fit.feature_importances_)
```

执行后，我们可以看见算法给出了每一个数据特征的得分。结果如下：

```
[ 0.12098909  0.2517376   0.10044005  0.08002211  0.07149687  0.12635934
  0.11878864  0.1301663 ]
```

9.6 总结

本章介绍了四种选定数据特征的方法。通过选定数据特征来训练算法，得到一个能够提高准确度的模型。得到的模型的好坏如何评估呢？接下来会介绍通过采样数据来评估算法模型的方法。

第四部分

选择模型

“所有模型都是坏的，但有些模型是有用的”。建立模型之后就要去评估模型，确定模型是否有用。模型评估是模型开发过程中不可或缺的一部分，有助于发现表达数据的最佳模型和所选模型将来工作的性能如何。

10 评估算法

要知道算法模型对未知的数据表现如何，最好的评估办法是利用已经明确知道结果的数据运行生成的算法模型进行验证。此外，还可以采用重新采样评估的方法，使用新的数据来评估算法模型。本章就将介绍如何使用 scikit-learn 中的采样评估办法来评价算法模型的准确度。

10.1 评估算法的方法

在评估机器学习算法的时候，为什么不将训练数据集直接作为评估数据集，最直接的原因是过度拟合，不能有效地发现算法模型的不足。所谓拟合是指已知某函数的若干离散函数值$\{f_1,f_2,\cdots,f_n\}$，通过调整该函数中若干待定系数$f(\lambda_1,\lambda_2,\cdots,\lambda_n)$，使该函数与已知点集的差别（最小二乘意义）最小。过度拟合是指为了得到一致假设变得过度严格。避免过度拟合是分类器设计中的一个核心任务，通常采用增大数据量和评估数据集的方法对分类器进行评估。

想象一下，假设算法记住了所有的训练数据集，当采用相同的数据集来评价算法时，会得到一个很高的评分。但是，当预测新数据集时或许会表现很糟糕。因此，必须使用

与训练数据集完全不同的评估数据集来评价算法。

评估就是估计算法在预测新数据的时候能达到什么程度，但这不是对算法准确度的保证。当评估完算法模型之后，可以用整个数据集（训练数据集和评估数据集的合集）重新训练算法，生成最终的算法模型。接下来将学习四种不同的分离数据集的方法，用来分离训练数据集和评估数据集，并用其评估算法模型：

- 分离训练数据集和评估数据集。
- K 折交叉验证分离。
- 弃一交叉验证分离。
- 重复随机评估、训练数据集分离。

10.2 分离训练数据集和评估数据集

最简单的方法就是将评估数据集和训练数据集完全分开，采用评估数据集来评估算法模型。可以简单地将原始数据集分为两部分，第一部分用来训练算法生成模型，第二部分通过模型来预测结果，并与已知的结果进行比较，来评估算法模型的准确度。如何分割数据集取决于数据集的规模，通常会将 67%的数据作为训练集，将 33%的数据作为评估数据集。

这是一种非常简洁，快速的数据分离技术，通常在具有大量数据、数据分布比较平衡，或者对问题的展示比较平均的情况下非常有效。这个方法非常快速，对某些执行比较慢的算法非常有效。下面给出一个简单的按照 67%:33%的比例分离数据，来评估逻辑回归模型的例子。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
```

```
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
test_size = 0.33
seed = 4
X_train, X_test, Y_traing, Y_test = train_test_split(X, Y, test_size=
test_size, random_state=seed)
model = LogisticRegression()
model.fit(X_train, Y_traing)
result = model.score(X_test, Y_test)
print("算法评估结果：%.3f%%" % (result * 100))
```

执行后得到的结果大约是 80%。需要注意的是，为了让算法模型具有良好的可复用性，在指定了分离数据的大小的同时，还指定了数据随机的粒度（seed=4），将数据随机进行分离。通过指定随机的粒度，可以确保每次执行程序得到相同的结果，这有助于比较两个不同的算法生成的模型的结果。为了保证算法比较是在相同的条件下执行的，必须保证训练数据集和评估数据集是相同的。执行结果如下：

```
算法评估结果：80.315%
```

10.3 K 折交叉验证分离

交叉验证是用来验证分类器的性能的一种统计分析方法，有时也称作循环估计，在统计学上是将数据样本切割成小子集的实用方法。基本思想是按照某种规则将原始数据进行分组，一部分作为训练数据集，另一部分作为评估数据集。首先用训练数据集对分类器进行训练，再利用评估数据集来测试训练得到的模型，以此作为评价分类器的性能指标。

K 折交叉验证是将原始数据分成 K 组（一般是均分），将每个子集数据分别做一次验证集，其余的 K-1 组子集数据作为训练集，这样会得到 K 个模型，再用这 K 个模型最终的验证集的分类准确率的平均数，作为此 K 折交叉验证下分类器的性能指标。K 一般大

于等于 2，实际操作时一般从 3 开始取值，只有在原始数据集和数据量小的时候才会尝试取 2。K 折交叉验证可以有效地避免过学习及欠学习状态的发生，最后得到的结果也比较具有说服力。通常情况下，K 的取值为 3、5、10。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)
model = LogisticRegression()
result = cross_val_score(model, X, Y, cv=kfold)
print("算法评估结果：%.3f%% (%.3f%%)" % (result.mean() * 100, result.std() *
100))
```

执行结果中给出了评估的得分及标准方差。执行结果如下：

```
算法评估结果：76.951% (4.841%)
```

10.4 弃一交叉验证分离

如果原始数据有 *N* 个样本，那么弃一交叉验证就是 *N*-1 个交叉验证，即每个样本单独作为验证集，其余的 *N*-1 个样本作为训练集，所以弃一交叉验证会得到 *N* 个模型，用这 *N* 个模型最终的验证集的分类准确率的平均数作为此次弃一交叉验证分类器的性能指标。相较于 K 折交叉验证，弃一交叉验证有两个显著的优点：

- 每一回合中几乎所有的样本皆用于训练模型，因此最接近原始样本的分布，这样评估所得的结果比较可靠。
- 实验过程中没有随机因素会影响实验数据，确保实验过程是可以被复制的。

但弃一交叉验证的缺点是计算成本高，因为需要建立的模型数量与原始数据样本数量相同，当原始数据样本数量相当多时，弃一交叉验证在实际运行上便有困难，需要花费大量的时间来完成算法的运算与评估， 除非每次训练分类器得到模型的速度很快，或者可以用并行化计算减少计算所需的时间。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
loocv = LeaveOneOut()
model = LogisticRegression()
result = cross_val_score(model, X, Y, cv=loocv)
print("算法评估结果：%.3f%% (%.3f%%)" % (result.mean() * 100, result.std() *
100))
```

利用此方法计算出的标准方差和 K 折交叉验证的结果有较大的差距。执行结果如下：

```
算法评估结果：76.823% (42.196%)
```

10.5 重复随机分离评估数据集与训练数据集

另外一种 K 折交叉验证的用途是随机分离数据为训练数据集和评估数据集，但是重复这个过程多次，就如同交叉验证分离。下面的例子就是将数据按照 67%:33%的比例分

离，然后重复这个过程 10 次。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
n_splits = 10
test_size = 0.33
seed = 7
kfold = ShuffleSplit(n_splits=n_splits, test_size=test_size,
random_state=seed)
model = LogisticRegression()
result = cross_val_score(model, X, Y, cv=kfold)
print("算法评估结果：%.3f%% (%.3f%%)" % (result.mean() * 100, result.std() *
100))
```

执行结果如下：

```
算法评估结果：76.496% (1.698%)
```

10.6 总结

本章介绍了四种用来分离数据集的方法，并通过分离后的评估数据集来评估算法模型。通常会按照下面的原则来选择数据分离的方法：

- K 折交叉验证是用来评估机器学习算法的黄金准则。通常会取 K 为 3、5、10 来分离数据。
- 分离训练数据集和评估数据集。因为执行效率比较高，通常会用于算法的执行效率比较低，或者具有大量数据的时候。

- 弃一交叉验证和重复随机分离评估数据集与训练数据集这两种方法，通常会用于平衡评估算法、模型训练的速度及数据集的大小。

还有一条黄金准则就是，当不知道如何选择分离数据集的方法时，请选择 K 折交叉验证来分离数据集；当不知道如何设定 K 值时，请将 K 值设为 10。

下一章将学习如何评估分类和回归算法的方法，并创建算法的评估报告，以便选择最终的算法和生成算法模型。

11 算法评估矩阵

如何来评估机器学习的算法模型是非常重要的。选择能够展示机器学习算法模型的准确度的评估矩阵，是计算和比较算法模型最好的方式。并且在评估算法时，计算并比较这些评估矩阵，可以快速地选择合适的算法。本章将学习如何使用 scikit-learn 来计算并展示算法的评估矩阵。

11.1 算法评估矩阵

本章将针对分类算法和回归算法介绍不同的评估矩阵。本章的例子中采用的数据全部来自于 UCI 机器学习仓库 。

- 分类算法矩阵将继续使用 Pima Indians 的数据集。这是一个二元分类问题，并且数据集的所有输入都是数字。
- 回归算法矩阵将使用波士顿房价（Boston House Price）数据集。这个数据集被广泛地用来介绍回归算法，并且数据集中所有的输入也都是数字。

在这里，分类算法矩阵以逻辑回归为例，回归算法矩阵以线性回归为例，使用实际应用中最常采用的 10 折交叉验证来分离数据，并计算展示算法的评估矩阵。

本章使用 scikit-learn 的 model_selection 中的 cross_val_score 方法来评估算法。这个方法不仅适用于分类算法，也适用于回归算法。更多的内容可以查阅 scikit-learn 的文档：Model evaluation:quantifying the quality of predictions。

11.2 分类算法矩阵

分类问题或许是最常见的机器学习问题，并且有多种评估矩阵来评估分类算法。本章将介绍以下几种用来评估分类算法的评估矩阵：

- 分类准确度。
- 对数损失函数（Logloss）。
- AUC 图。
- 混淆矩阵。
- 分类报告（Classification Report）。

11.2.1 分类准确度

分类准确度就是算法自动分类正确的样本数除以所有的样本数得出的结果。通常，准确度越高，分类器越好。这是分类算法中最常见，也最易被误用的评估参数。准确度确实是一个很好、很直观的评价指标，但是有时候准确度高并不代表算法就一定好。比如对某个地区某天地震的预测，假设有一堆的特征作为地震分类的属性，类别却只有两个（0：不发生地震，1：发生地震）。一个不加思考的分类器对每一个测试用例都将类别划分为 0，那它就可能达到 99%的准确度，但真的地震时，这个分类器却毫无察觉，这个分类器造成的损失是巨大的。为什么拥有 99%的准确度的分类器却不是我们想要的，因为数据分布不均衡，类别 1 的数据太少，完全错分类别 1 依然可以达到很高的准确度，却忽视了需要关注的事实和现象。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
```

```
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)
model = LogisticRegression()
result = cross_val_score(model, X, Y, cv=kfold)
print("算法评估结果准确度：%.3f (%.3f)" % (result.mean(), result.std()))
```

执行结果如下：

```
算法评估结果准确度：0.770 (0.048)
```

11.2.2 对数损失函数

在逻辑回归的推导中，它假设样本服从伯努利分布（0~1 分布），然后求得满足该分布的似然函数，再取对数、求极值等。而逻辑回归并没有求似然函数的极值，而是把极大化当作一种思想，进而推导出它的经验风险函数为：最小化负的似然函数[max $F(y, f(x)) \rightarrow \min -F(y, f(x))$]。从损失函数的视角来看，它就成了对数（Log）损失函数了。对数损失函数越小，模型就越好，而且使损失函数尽量是一个凸函数，便于收敛计算。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
```

```
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)
model = LogisticRegression()
scoring = 'neg_log_loss'
result = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print('Logloss %.3f (%.3f)' % (result.mean(), result.std()))
```

执行结果如下：

```
Logloss -0.493 (0.047)
```

11.2.3 AUC 图

ROC 和 AUC 是评价分类器的指标。ROC 是受试者工作特征曲线（Receiver Operating Characteristic Curve）的简写，又称为感受性曲线（Sensitivity Curve）。得此名的原因在于曲线上各点反映相同的感受性，它们都是对同一信号刺激的反应，只不过是在几种不同的判定标准下所得的结果而已。ROC 是反映敏感性和特异性连续变量的综合指标，用构图法揭示敏感性和特异性的相互关系，通过将连续变量设定出多个不同的临界值计算出一系列敏感性和特异性，再以敏感性为纵坐标、（1-特异性）为横坐标绘制成曲线。AUC 是 ROC 曲线下的面积（Area Under ROC Curve）的简称，顾名思义，AUC 的值就是处于 ROC Curve 下方的那部分面积的大小。通常，AUC 的值介于 0.5 到 1.0 之间，AUC 的值越大，诊断准确性越高。在 ROC 曲线上，靠近坐标图左上方的点为敏感性和特异性均较高的临界值。

为了解释 ROC 的概念，让我们考虑一个二分类问题，即将实例分成正类（Positive）或负类（Negative）。对一个二分类问题来说，会出现四种情况：如果一个实例是正类并且也被预测成正类，即为真正类（True Positive）；如果实例是负类却被预测成正类，称之为假正类（False Positive）。相应地，如果实例是负类也被预测成负类，称之为真负类

（True Negative）；如果实例为正类却被预测成负类，则为假负类（False Negative）。列联表或混淆矩阵如表 11-1 所示，“1”代表正类，“0”代表负类。

表 11-1

		实　际	
		1	0
预测	1	True Positive (TP) 真正	False Positive (FP) 假正
	0	False Negative (FN) 假负	True Negative TN 真负

基于该列联表，定义敏感性指标为：sensitivity=TP/(TP+FN)。敏感性指标又称为真正类率（True Positive Rate ，TPR），计算的是分类器所识别出的正实例占所有正实例的比例。

另外，定义负正类率（False Positive Rate，FPR）的计算公式为：FPR=FP/(FP+TN)，负正类率计算的是分类器错认为正类的负实例占所有负实例的比例。

定义特异性指标为：Specificity=TN/(FP+TN)=1−FPR。特异性指标又称为真负类率（True Negative Rate，TNR）。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)
```

```
model = LogisticRegression()
scoring = 'roc_auc'
result = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print('AUC %.3f (%.3f)' % (result.mean(), result.std()))
```

执行结果如下：

```
AUC 0.824 (0.041)
```

11.2.4 混淆矩阵

混淆矩阵（Cnfusion Matrix）主要用于比较分类结果和实际测得值，可以把分类结果的精度显示在一个混淆矩阵里面。混淆矩阵是可视化工具，特别适用于监督学习，在无监督学习时一般叫作匹配矩阵。混淆矩阵的每列代表预测类别，每列的总数表示预测为该类别的数据的数目；每行代表数据的真实归属类别，每行的数据总数表示该类别的数据的数目。每列中的数值表示真实数据被预测为该类的数目。如表 11-2 所示，有 150 个样本数据，这些数据分成 3 类，每类 50 个。分类结束后得到的混淆矩阵，每行之和为 50，表示 50 个样本。第一行说明类 1 的 50 个样本有 43 个分类正确，5 个错分为类 2，2 个错分为类 3。

表 11-2

		预　测		
		类 1	类 2	类 3
实际	类 1	43	5	2
	类 2	2	45	3
	类 3	0	1	49

代码如下：

```
from pandas import read_csv
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
# 导入数据
```

```
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
test_size = 0.33
seed = 4
X_train,    X_test,    Y_traing,    Y_test    =    train_test_split(X,    Y,
test_size=test_size, random_state=seed)
model = LogisticRegression()
model.fit(X_train, Y_traing)
predicted = model.predict(X_test)
matrix = confusion_matrix(Y_test, predicted)
classes = ['0', '1']
dataframe = pd.DataFrame(data=matrix,
                    index=classes,
                    columns=classes)
print(dataframe)
```

执行结果如下：

```
     0   1
 0  152  19
 1   31  52
```

11.2.5 分类报告

在 scikit-learn 中提供了一个非常方便的工具，可以给出对分类问题的评估报告，Classification_report()方法能够给出精确率(precision)、召回率(recall)、F1 值(F1-score)和样本数目 (support)。在这里简单地介绍一下三个指标数据：精确率、召回率、F1 值。

在介绍这三个指标数据之前，我们先定义 TP、FN、FP、TN 四种分类情况，我们借助表 11-3 来说明。

表 11-3

		实　际	
		1	0
预测	1	True Positive（TP） 真正类	False Positive（FP） 负正类
	0	False Negative（F） 假负类	True Negative TN 真负类

精确率的公式 P=TP/(TP+FP)，计算的是所有被检索到的项目中应该被检索到的项目占的比例。

召回率的公式 R=TP/(TP+FN)，计算的是所有检索到的项目占所有应该检索到的项目的比例。

$F1$ 值就是精确率和召回率的调和均值，也就是 $2F1=P+R$ 。

代码如下：

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
test_size = 0.33
seed = 4
X_train,    X_test,    Y_traing,    Y_test    =    train_test_split(X,    Y,
test_size=test_size, random_state=seed)
model = LogisticRegression()
model.fit(X_train, Y_traing)
predicted = model.predict(X_test)
```

```
report = classification_report(Y_test, predicted)
print(report)
```

执行结果如下：

```
             precision    recall  f1-score   support

        0.0       0.83       0.89      0.86       171
        1.0       0.73       0.63      0.68        83

avg / total       0.80       0.80      0.80       254
```

11.3 回归算法矩阵

接下来将介绍三种评估机器学习的回归算法的评估矩阵。

- 平均绝对误差（Mean Absolute Error，MAE）。
- 均方误差（Mean Squared Error，MSE）。
- 决定系数（R^2）。

11.3.1 平均绝对误差

平均绝对误差是所有单个观测值与算术平均值的偏差的绝对值的平均值。与平均误差相比，平均绝对误差由于离差被绝对值化，不会出现正负相抵消的情况，因而，平均绝对误差能更好地反映预测值误差的实际情况。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
# 导入数据
filename = 'housing.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
        'RAD', 'TAX', 'PRTATIO', 'B', 'LSTAT', 'MEDV']
data = read_csv(filename, names=names, delim_whitespace=True)
# 将数据分为输入数据和输出结果
array = data.values
```

```
X = array[:, 0:13]
Y = array[:, 13]
n_splits = 10
seed = 7
kfold = KFold(n_splits=n_splits, random_state=seed)
model = LinearRegression()
scoring = 'neg_mean_absolute_error'
result = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print('MAE: %.3f (%.3f)' % (result.mean(), result.std()))
```

执行结果如下：

```
MAE: -4.005 (2.084)
```

11.3.2 均方误差

均方误差是衡量平均误差的方法，可以评价数据的变化程度。均方根误差是均方误差的算术平方根。均方误差的值越小，说明用该预测模型描述实验数据的准确度越高。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
# 导入数据
filename = 'housing.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
        'RAD', 'TAX', 'PRTATIO', 'B', 'LSTAT', 'MEDV']
data = read_csv(filename, names=names, delim_whitespace=True)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:13]
Y = array[:, 13]
n_splits = 10
seed = 7
kfold = KFold(n_splits=n_splits, random_state=seed)
model = LinearRegression()
scoring = 'neg_mean_squared_error'
```

```
result = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print('MSE: %.3f (%.3f)' % (result.mean(), result.std()))
```

执行结果如下：

```
MSE: -34.705 (45.574)
```

11.3.3 决定系数（R^2）

决定系数，反映因变量的全部变异能通过回归关系被自变量解释的比例。拟合优度越大，自变量对因变量的解释程度越高，自变量引起的变动占总变动的百分比越高，观察点在回归直线附近越密集。如R^2为 0.8，则表示回归关系可以解释因变量 80%的变异。换句话说，如果我们能控制自变量不变，则因变量的变异程度会减少 80%。

决定系数（R^2）的特点：

- 可决系数是非负的统计量。
- 可决系数的取值范围：$0 \leqslant R^2 \leqslant 1$。
- 可决系数是样本观测值的函数，是因随机抽样而变动的随机变量。为此，对可决系数的统计的可靠性也应进行检验。

代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
# 导入数据
filename = 'housing.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
        'RAD', 'TAX', 'PRTATIO', 'B', 'LSTAT', 'MEDV']
data = read_csv(filename, names=names, delim_whitespace=True)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:13]
Y = array[:, 13]
n_splits = 10
```

```
seed = 7
kfold = KFold(n_splits=n_splits, random_state=seed)
model = LinearRegression()
scoring = 'r2'
result = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print('R2: %.3f (%.3f)' % (result.mean(), result.std()))
```

执行结果如下：

```
R2: 0.203 (0.595)
```

11.4 总结

本章分别介绍了分类算法和回归算法的评估矩阵，并实践了 scikit-learn 中提供的计算评估矩阵的方法。下一章将介绍如何通过抽样分析来分析算法自身，并选择合适的算法。

12 审查分类算法

算法审查是选择合适的机器学习算法的主要方法之一。审查算法前并不知道哪个算法对问题最有效，必须设计一定的实验进行验证，以找到对问题最有效的算法。本章将学习通过 scikit-learn 来审查六种机器学习的分类算法，通过比较算法评估矩阵的结果，选择合适的算法。本章将学到：

- 如何审查机器学习的分类算法。
- 如何审查两个线性分类算法。
- 如何审查四个非线性分类算法。

12.1 算法审查

审查算法前没有办法判断哪个算法对数据集最有效、能够生成最优模型，必须通过一系列实验判断出哪些算法对问题最有效，然后再进一步来选择算法。这个过程被叫作算法审查。

在选择算法时，应该换一种思路，不是针对数据应该采用哪种算法，而是应该用数据来审查哪些算法。应该先猜测一下，什么算法会具有最好的效果。这是训练我们对数

据敏感性的好方法。我非常建议大家对同一个数据集运用不同的算法，来审查算法的有效性，然后找到最有效的算法。下面是审查算法的几点建议：

- 尝试多种代表性算法。
- 尝试多种机器学习的算法。
- 尝试多种模型。

接下来会介绍几种常见的分类算法。

12.2 算法概述

在分类算法中，目前存在很多类型的分类器：线性分类器、贝叶斯分类器、基于距离的分类器等。接下来会介绍六种分类算法，先介绍两种线性算法：

- 逻辑回归。
- 线性判别分析。

再介绍四种非线性算法：

- K 近邻。
- 贝叶斯分类器。
- 分类与回归树。
- 支持向量机。

本章继续使用 Pima Indians 数据集来审查算法，同时会采用 10 折交叉验证来评估算法的准确度。使用平均准确度来标准化算法的得分，以减少数据分布不均衡对算法的影响。

12.3 线性算法

逻辑回归和线性判别分析都是假定输入的数据符合高斯分布。

12.3.1 逻辑回归

回归是一种极易理解的模型，相当于 $y=f(x)$，表明自变量 x 与因变量 y 的关系。犹如医生治病时先望、闻、问、切，再判定病人是否生病或生了什么病，此处的“望、闻、问、切”就是获取自变量 x，即特征数据；判断是否生病就相当于获取因变量 y，即预测分类。逻辑回归其实是一个分类算法而不是回归算法，通常是利用已知的自变量来预测一个离散型因变量的值（如二进制值 0/1、是/否、真/假）。简单来说，它就是通过拟合一个逻辑函数（Logit Function）来预测一个事件发生的概率。所以它预测的是一个概率值，它的输出值应该为 0~1，因此非常适合处理二分类问题。在 scikit-learn 中的实现类是 LogisticRegression。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)
model = LogisticRegression()
result = cross_val_score(model, X, Y, cv=kfold)
print(result.mean())
```

执行结果如下：

```
0.76951469583
```

12.3.2 线性判别分析

线性判别分析（Linear Discriminant Analysis，LDA），也叫作 Fisher 线性判别（Fisher Linear Discriminant，FLD），是模式识别的经典算法，它是在 1996 年由 Belhumeur 引入模式识别和人工智能领域的。线性判别分析的基本思想是将高维的模式样本投影到最佳鉴别矢量空间，以达到抽取分类信息和压缩特征空间维数的效果，投影后保证模式样本在新的子空间有最大的类间距离和最小的类内距离，即模式在该空间中有最佳的可分离性。因此，它是一种有效的特征抽取方法。使用这种方法能够使投影后模式样本的类间散布矩阵最大，并且类内散布矩阵最小。就是说，它能够保证投影后模式样本在新的空间中有最小的类内距离和最大的类间距离，即模式在该空间中有最佳的可分离性。线性判别分析与主要成分分析一样，被广泛应用在数据降维中。在 scikit-learn 中的实现类是 LinearDiscriminantAnalysis。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)
model = LinearDiscriminantAnalysis()
result = cross_val_score(model, X, Y, cv=kfold)
print(result.mean())
```

执行结果如下：

```
0.773462064252
```

12.4 非线性算法

下面介绍四种非线性算法：K 近邻（KNN）、贝叶斯分类器、分类与回归树和支持向量机算法。

12.4.1 K 近邻算法

K 近邻算法是一种理论上比较成熟的方法，也是最简单的机器学习算法之一。该方法的思路是：如果一个样本在特征空间中的 k 个最相似（特征空间中最邻近）的样本中的大多数属于某一个类别，则该样本也属于这个类别。在 KNN 中，通过计算对象间距离来作为各个对象之间的非相似性指标，避免了对象之间的匹配问题，距离一般使用欧氏距离或曼哈顿距离；同时，KNN 通过依据 k 个对象中占优的类别进行决策，而不是通过单一的对象类别决策。这就是 KNN 算法的优势。在 scikit-learn 中的实现类是 KNeighborsClassifier。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)
model = KNeighborsClassifier()
result = cross_val_score(model, X, Y, cv=kfold)
print(result.mean())
```

执行结果如下：

```
0.726555023923
```

12.4.2 贝叶斯分类器

贝叶斯分类器的分类原理是通过某对象的先验概率，利用贝叶斯公式计算出其在所有类别上的后验概率，即该对象属于某一类的概率，选择具有最大后验概率的类作为该对象所属的类。也就是说，贝叶斯分类器是最小错误率意义上的优化。对于给出的待分类项，求解在此项出现的条件下各个类别出现的概率，哪个最大就认为此待分类项属于哪个类别。贝叶斯分类器的特点如下：

- 贝叶斯分类器是一种基于统计的分类器，它根据给定样本属于某一个具体类的概率来对其进行分类。
- 贝叶斯分类器的理论基础是贝叶斯理论。
- 贝叶斯分类器的一种简单形式是朴素贝叶斯分类器，与随机森林、神经网络等分类器都具有可比的性能。
- 贝叶斯分类器是一种增量型的分类器。

在贝叶斯分类器中，对输入数据同样做了符合高斯分布的假设。在 scikit-learn 中的实现类是 GaussianNB。

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import GaussianNB
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)
model = GaussianNB()
result = cross_val_score(model, X, Y, cv=kfold)
print(result.mean())
```

执行结果如下：

```
0.75517771702
```

12.4.3 分类与回归树

分类与回归树的英文缩写是 CART，也属于一种决策树，树的构建基于基尼指数。CART 假设决策树是二叉树，内部结点特征的取值为“是”和“否”，左分支是取值为“是”的分支，右分支是取值为“否”的分支。这样的决策树等价于递归二分每个特征，将输入空间（特征空间）划分为有限个单元，并在这些单元上确定预测的概率分布，也就是在输入给定的条件下输出的条件概率分布。CART 算法由以下两步组成。

- 树的生成：基于训练数据集生成决策树，生成的决策树要尽量大。
- 树的剪枝：用验证数据集对已生成的树进行剪枝，并选择最优子树，这时以损失函数最小作为剪枝的标准。

决策树的生成就是通过递归构建二叉决策树的过程，对回归树用平方误差最小化准则，或对分类树用基尼指数最小化准则，进行特征选择，生成二叉树。可以通过 scikit-learn 中的 DecisionTreeClassifier 类来构建一个 CART 模型。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)
model = DecisionTreeClassifier()
```

```
result = cross_val_score(model, X, Y, cv=kfold)
print(result.mean())
```

执行结果如下：

```
0.701708817498
```

12.4.4 支持向量机

支持向量机是 Corinna Cortes 和 Vapnik 等于 1995 年首先提出的，它在解决小样本、非线性及高维模式识别中表现出许多特有的优势，并能够推广应用到函数拟合等其他机器学习问题中。在机器学习中，支持向量机（SVM）是与相关的学习算法有关的监督学习模型，可以分析数据、识别模式，用于分类和回归分析。给定一组训练样本，每条记录标记所属类别，使用支持向量机算法进行训练，并建立一个模型，对新数据实例进行分类，使其成为非概率二元线性分类。一个 SVM 模型的例子是，如在空间中的不同点的映射，使得所属不同类别的实例是由一个差距明显且尽可能宽的划分表示。新的实例则映射到相同的空间中，并基于它们落在相同间隙上预测其属于同一个类别。现在 SVM 也被扩展到处理多分类问题，可以通过 scikit-learn 中的 SVC 类来构建一个 SVM 模型。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)
model = SVC()
```

```
result = cross_val_score(model, X, Y, cv=kfold)
print(result.mean())
```

执行结果如下：

```
0.651025290499
```

12.5 总结

本章介绍了六种分类算法，以及它们在 scikit-learn 中的实现。算法主要分为：线性算法、距离算法、树算法、统计算法等。每一种算法都有不同的适用场景，对数据集有不同的要求。本章利用 Pima Indians 数据集对这几种算法进行了审查，这是选择合适的算法模型的有效方法。下一章将介绍如何审查回归算法，以处理回归问题。

13 审查回归算法

上一章介绍了如何审查分类算法，并介绍了六种不同的分类算法，还用同一个数据集按照相同的方式对它们做了审查，本章将用相同的方式对回归算法进行审查。在本章将学到：

- 如何审查机器学习的回归算法。
- 如何审查四种线性分类算法。
- 如何审查三种非线性分类算法。

13.1 算法概述

本章将审查七种回归算法。首先介绍四种线性算法：

- 线性回归算法。
- 岭回归算法（脊回归算法）。
- 套索回归算法。
- 弹性网络（Elastic Net）回归算法。

然后介绍三种非线性算法：

- K 近邻算法（KNN）。
- 分类与回归树算法。
- 支持向量机（SVM）。

本章将使用波士顿房价的数据集来审查回归算法，采用 10 折交叉验证来分离数据，并应用到所有的算法上。另外，还会通过均方误差来评估算法模型。scikit-learn 中的 cross_val_score()函数能够帮助评估算法模型，我们就用这个函数来评估算法模型。

13.2 线性算法

首先介绍 scikit-learn 中用来处理机器学习中的回归问题的四种算法。

13.2.1 线性回归算法

线性回归算法是利用数理统计中的回归分析，来确定两种或两种以上变量间相互依赖的定量关系的一种统计分析方法，运用十分广泛。其表达式为 $y = w'x+e$，e 表示误差服从均值为 0 的正态分布。在回归分析中，只包括一个自变量和一个因变量，且二者的关系可用一条直线近似表示，这种回归分析称为一元线性回归分析。如果回归分析中包括两个或两个以上的自变量，且因变量和自变量之间是线性关系，则称为多元线性回归分析。在 scikit-learn 中实现线性回归算法的是 LinearRegression 类。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
# 导入数据
filename = 'housing.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
        'RAD', 'TAX', 'PRTATIO', 'B', 'LSTAT', 'MEDV']
data = read_csv(filename, names=names, delim_whitespace=True)
# 将数据分为输入数据和输出结果
array = data.values
```

```
X = array[:, 0:13]
Y = array[:, 13]
n_splits = 10
seed = 7
kfold = KFold(n_splits=n_splits, random_state=seed)
model = LinearRegression()
scoring = 'neg_mean_squared_error'
result = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print('Linear Regression: %.3f' % result.mean())
```

执行结果如下：

```
Linear Regression: -34.705
```

13.2.2 岭回归算法

岭回归算法是一种专门用于共线性数据分析的有偏估计回归方法，实际上是一种改良的最小二乘估计法，通过放弃最小二乘法的无偏性，以损失部分信息、降低精度为代价，获得回归系数更符合实际、更可靠的回归方法，对病态数据的拟合要强于最小二乘法。在 scikit-learn 中实现岭回归算法的是 Ridge 类。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import Ridge
# 导入数据
filename = 'housing.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
        'RAD', 'TAX', 'PRTATIO', 'B', 'LSTAT', 'MEDV']
data = read_csv(filename, names=names, delim_whitespace=True)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:13]
Y = array[:, 13]
n_splits = 10
seed = 7
kfold = KFold(n_splits=n_splits, random_state=seed)
model = Ridge()
```

```
scoring = 'neg_mean_squared_error'
result = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print('Ridge Regression: %.3f' % result.mean())
```

执行结果如下：

```
Ridge Regression: -34.078
```

13.2.3 套索回归算法

套索回归算法和岭回归算法类似，套索回归算法也会惩罚回归系数，在套索回归中会惩罚回归系数的绝对值大小。此外，它能够减少变化程度并提高线性回归模型的精度。套索回归算法和岭回归算法有一点不同，它使用的惩罚函数是绝对值，而不是平方。这导致惩罚（或等于约束估计的绝对值之和）值使一些参数估计结果等于零。使用惩罚值越大，进一步估计会使缩小值越趋近零。这将导致我们要从给定的 n 个变量中选择变量。如果预测的一组变量高度相似，套索回归算法会选择其中的一个变量，并将其他的变量收缩为零。在 scikit-learn 中的实现类是 Lasso。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import Lasso
# 导入数据
filename = 'housing.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
        'RAD', 'TAX', 'PRTATIO', 'B', 'LSTAT', 'MEDV']
data = read_csv(filename, names=names, delim_whitespace=True)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:13]
Y = array[:, 13]
n_splits = 10
seed = 7
kfold = KFold(n_splits=n_splits, random_state=seed)
model = Lasso()
scoring = 'neg_mean_squared_error'
```

```
result = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print('Lasso Regression: %.3f' % result.mean())
```

执行结果如下：

```
Lasso Regression: -34.464
```

13.2.4 弹性网络回归算法

弹性网络回归算法是套索回归算法和岭回归算法的混合体，在模型训练时，弹性网络回归算法综合使用 L1 和 L2 两种正则化方法。当有多个相关的特征时，弹性网络回归算法是很有用的，套索回归算法会随机挑选算法中的一个，而弹性网络回归算法则会选择两个。与套索回归算法和岭回归算法相比，弹性网络回归算法的优点是，它允许弹性网络回归继承循环状态下岭回归的一些稳定性。另外，在高度相关变量的情况下，它会产生群体效应；选择变量的数目没有限制；可以承受双重收缩。在 scikit-learn 中的实现类是 ElasticNet。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import ElasticNet
# 导入数据
filename = 'housing.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
        'RAD', 'TAX', 'PRTATIO', 'B', 'LSTAT', 'MEDV']
data = read_csv(filename, names=names, delim_whitespace=True)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:13]
Y = array[:, 13]
n_splits = 10
seed = 7
kfold = KFold(n_splits=n_splits, random_state=seed)
model = ElasticNet()
scoring = 'neg_mean_squared_error'
result = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print('ElasticNet Regression: %.3f' % result.mean())
```

执行结果如下：

```
ElasticNet Regression: -31.165
```

13.3 非线性算法

接下来我们将介绍 scikit-learn 中机器学习的三种非线性回归算法。这三种算法在分类算法中也存在，所以就不对算法做重复的介绍，只介绍它们在 scikit-learn 中的实现类。

13.3.1 K 近邻算法

K 近邻算法是按照距离来预测结果。在 scikit-learn 中对回归算法的 K 近邻算法的实现类是 KNeighborsRegressor。默认的距离参数为闵式距离，可以指定曼哈顿距离作为距离的计算方式。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsRegressor
# 导入数据
filename = 'housing.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
        'RAD', 'TAX', 'PRTATIO', 'B', 'LSTAT', 'MEDV']
data = read_csv(filename, names=names, delim_whitespace=True)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:13]
Y = array[:, 13]
n_splits = 10
seed = 7
kfold = KFold(n_splits=n_splits, random_state=seed)
model = KNeighborsRegressor()
scoring = 'neg_mean_squared_error'
result = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print('KNeighbors Regression: %.3f' % result.mean())
```

执行结果如下：

```
KNeighbors Regression: -107.287
```

13.3.2 分类与回归树

在上一章中已经介绍过分类与回归树的算法，它同样适用于回归问题。在 scikit-learn 中分类与回归树的实现类是 DecisionTreeRegressor。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeRegressor
# 导入数据
filename = 'housing.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
        'RAD', 'TAX', 'PRTATIO', 'B', 'LSTAT', 'MEDV']
data = read_csv(filename, names=names, delim_whitespace=True)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:13]
Y = array[:, 13]
n_splits = 10
seed = 7
kfold = KFold(n_splits=n_splits, random_state=seed)
model = DecisionTreeRegressor()
scoring = 'neg_mean_squared_error'
result = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print('CART: %.3f' % result.mean())
```

执行结果如下：

```
CART: -37.560
```

13.3.3 支持向量机

支持向量机的算法在上一章也介绍过了，它同样可以用来处理回归问题。在 scikit-learn 中处理回归问题的实现类是 SVR。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVR
# 导入数据
filename = 'housing.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
        'RAD', 'TAX', 'PRTATIO', 'B', 'LSTAT', 'MEDV']
data = read_csv(filename, names=names, delim_whitespace=True)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:13]
Y = array[:, 13]
n_splits = 10
seed = 7
kfold = KFold(n_splits=n_splits, random_state=seed)
model = SVR()
scoring = 'neg_mean_squared_error'
result = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print('SVM: %.3f' % result.mean())
```

执行结果如下：

```
SVM: -91.048
```

13.4 总结

在 scikit-learn 中，算法评估矩阵的计算经常使用 cross_val_score 函数，通过指定 scoring 参数来选择使用的不同评估矩阵。scoring 的参数（摘自 scikit-learn 的使用指南）如表 13-1 所示。

表 13-1

Scoring	Function	Comment
Classification		
Accuracy	metrics.accuracy_score	
average_precision	metrics.average_precision_score	

续表

Scoring	Function	Comment
f1	metrics.f1_score	for binary targets
f1_micro	metrics.f1_score	micro-averaged
f1_macro	metrics.f1_score	macro-averaged
f1_weighted	metrics.f1_score	weighted average
f1_samples	metrics.f1_score	by multilabel sample
neg_log_loss	metrics.log_loss	requires predict_proba support
precision etc.	metrics.precision_score	suffixes apply as with f1
recall etc.	metrics.recall_score	suffixes apply as with f1
roc_auc	metrics.roc_auc_score	
Clustering		
adjusted_mutual_info_score	metrics.adjusted_mutual_info_score	
adjusted_rand_score	metrics.adjusted_rand_score	
completeness_score	metrics.completeness_score	
fowlkes_mallows_score	metrics.fowlkes_mallows_score	
homogeneity_score	metrics.homogeneity_score	
mutual_info_score	metrics.mutual_info_score	
normalized_mutual_info_score	metrics.normalized_mutual_info_score	
v_measure_score	metrics.v_measure_score	
Regression		
explained_variance	metrics.explained_variance_score	
neg_mean_absolute_error	metrics.mean_absolute_error	
neg_mean_squared_error	metrics.mean_squared_error	
neg_mean_squared_log_error	metrics.mean_squared_log_error	
neg_median_absolute_error	metrics.median_absolute_error	
r2	metrics.r2_score	

本章对回归算法进行了审查，使用同一个数据集评估了不同的回归算法，这为处理回归问题进行算法选择时提供了有效的方法 。本章介绍了四种线性算法和三种非线性算法，并了解了如何比较选择回归算法。下一章将学习如何通过设计实验选择合适的算法。

14 算法比较

比较不同算法的准确度，选择合适的算法，在处理机器学习的问题时是非常重要的。本章将介绍一种模式，在 scikit-learn 中可以利用它比较不同的算法，并选择合适的算法。你可以将这种模式作为自己的模板，来处理机器学习的问题；也可以通过对其他不同算法的比较，改进这个模板。在本章将会学习以下内容：

- 如何设计一个实验来比较不同的机器学习算法。
- 一个可以重复利用的、用来评估算法性能的模板。
- 如何可视化算法的比较结果。

14.1 选择最佳的机器学习算法

当参与一个机器学习的项目时，会经常因为如何选择一种合适的算法模型而苦恼。每种模型都有各自适合处理的数据特征，通过交叉验证等抽样验证方式可以得到每种模型的准确度，并选择合适的算法。通过这种评估方式，可以找到一种或两种最适合问题的算法 。

当得到一个新的数据集时，应该通过不同的维度来审查数据，以便于找到数据的特征，这种方法也适用于选择算法模型。同样需要从不同的维度，用不同的方法来观察机器学习算法的准确度，并从中选择一种或两种对问题最有效的算法。一种比较好的方法是通过可视化的方式来展示平均准确度、方差等属性，以便于更方便地选择算法。接下来就介绍如何通过 scikit-learn 来实现对算法的比较。

14.2 机器学习算法的比较

最合适的算法比较方法是：使用相同的数据、相同的方法来评估不同的算法，以便得到一个准确的结果。下面将使用同一个数据集来比较六种分类算法，以便选择合适的算法来解决问题。

- 逻辑回归（LR）。
- 线性判别分析（LDA）。
- K 近邻（KNN）。
- 分类与回归树（CART）。
- 贝叶斯分类器。
- 支持向量机（SVM）。

我们继续使用 Pima Indians 数据集来介绍如何比较算法。这个数据集是一个二分类数据集，结果只有两个分类；用来训练算法模型的数据是八种全部由数字构成的属性特征值。采用 10 折交叉验证来分离数据，并采用相同的随机数分配方式来确保所有的算法都使用相同的数据。为了便于整理结果，给每一种算法设定一个短名字。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
```

```
from sklearn.naive_bayes import GaussianNB
from matplotlib import pyplot
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)
models = {}
models['LR'] = LogisticRegression()
models['LDA'] = LinearDiscriminantAnalysis()
models['KNN'] = KNeighborsClassifier()
models['CART'] = DecisionTreeClassifier()
models['SVM'] = SVC()
models['NB'] = GaussianNB()
results = []
for name in models:
    result = cross_val_score(models[name], X, Y, cv=kfold)
    results.append(result)
    msg = '%s: %.3f (%.3f)' % (name, result.mean(), result.std())
    print(msg)

# 图表显示
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(models.keys())
pyplot.show()
```

执行结果给出了每种算法的平均准确度和标准方差。结果如下：

```
LR: 0.770 (0.048)
LDA: 0.773 (0.052)
```

```
KNN: 0.727 (0.062)
CART: 0.694 (0.068)
SVM: 0.651 (0.072)
NB: 0.755 (0.043)
```

同时也可以通过箱线图展示算法的准确度，以及 10 折交叉验证中每次验证结果的分布状况。其执行结果如图 14-1 所示。

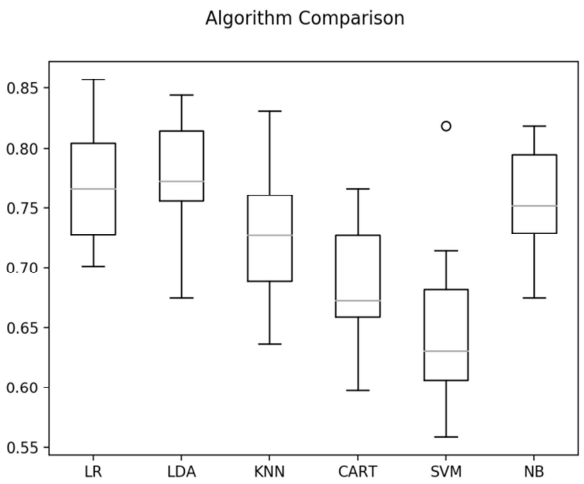


图 14-1

14.3 总结

本章给出了一种对多种算法进行分析比较的方法。通过这个方法可以找到一种或两种算法对给定数据集能够生成准确度最高的模型，从而选择合适的算法。这个方法也可以应用到所有机器学习的问题中。接下来将学习在 scikit-learn 中如何通过 Pipelines 来实现自动化流程处理。

15 自动流程

有一些标准的流程可以实现对机器学习问题的自动化处理，在 scikit-learn 中通过 Pipeline 来定义和自动化运行这些流程。本章就将介绍如何通过 Pipeline 实现自动化流程处理。本章包含以下内容：

- 如何通过 Pipeline 来最小化数据缺失。
- 如何构建数据准备和生成模型的 Pipeline。
- 如何构建特征选择和生成模型的 Pipeline。

15.1 机器学习的自动流程

在机器学习方面有一些可以采用的标准化流程，这些标准化流程是从共同的问题中提炼出来的，例如评估框架中的数据缺失等。在 scikit-learn 中提供了自动化运行流程的工具——Pipeline。Pipeline 能够将从数据转换到评估模型的整个机器学习流程进行自动化处理。读者可以到 scikit-learn 的官方网站阅读关于 Pipeline 的章节，加深对 Pipeline 的理解。

15.2 数据准备和生成模型的 Pipeline

在机器学习的实践中有一个很常见的错误，就是训练数据集与评估数据集之间的数据泄露，这会影响到评估的准确度。要想避免这个问题，需要有一个合适的方式把数据分离成训练数据集和评估数据集，这个过程被包含在数据的准备过程中。数据准备过程是很好的理解数据和算法关系的过程，举例来说，当对训练数据集做标准化和正态化处理来训练算法时，就应该理解并接受这同样要受评估数据集的影响。

Pipeline 能够处理训练数据集与评估数据集之间的数据泄露问题，通常会在数据处理过程中对分离出的所有数据子集做同样的数据处理，如正态化处理。下面将演示如何通过 Pipeline 来处理这个过程，共分为以下两个步骤：

（1）正态化数据。

（2）训练一个线性判别分析模型。

在使用 Pipeline 进行流程化算法模型的评估过程中，采用 10 折交叉验证来分离数据集。其代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import  Pipeline
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)
```

```
steps = []
steps.append(('Standardize', StandardScaler()))
steps.append(('lda', LinearDiscriminantAnalysis()))
model = Pipeline(steps)
result = cross_val_score(model, X, Y, cv=kfold)
print(result.mean())
```

Pipeline 的各个步骤，通过列表参数传递给 Pipeline 实例，并通过 Pipeline 进行流程化处理过程。执行结果如下：

```
0.773462064252
```

15.3 特征选择和生成模型的 Pipeline

特征选择也是一个容易受到数据泄露影响的过程。和数据准备一样，特征选择时也必须确保数据的稳固性，Pipeline 也提供了一个工具（FeatureUnion）来保证数据特征选择时数据的稳固性。下面是一个在数据选择过程中保持数据稳固性的例子。这个过程包括以下四个步骤：

（1）通过主要成分分析进行特征选择。

（2）通过统计选择进行特征选择。

（3）特征集合。

（4）生成一个逻辑回归模型。

在本例中也采用 10 折交叉验证来分离训练数据集和评估数据集。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import FeatureUnion
from sklearn.pipeline import  Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest
```

```
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)

# 生成 FeatureUnion
features = []
features.append(('pca', PCA()))
features.append(('select_best', SelectKBest(k=6)))
# 生成 Pipeline
steps = []
steps.append(('feature_union', FeatureUnion(features)))
steps.append(('logistic', LogisticRegression()))
model = Pipeline(steps)
result = cross_val_score(model, X, Y, cv=kfold)
print(result.mean())
```

此处先创建了FeatureUnion，然后将其作为Pipeline的一个生成步骤。执行结果如下：

```
0.779955570745
```

15.4 总结

本章学习了通过 scikit-learn 中的 Pipeline 进行自动流程化数据准备和特征选择的过程。接下来将探讨针对要处理的问题，如何提高机器学习算法的准确度。

第五部分

优化模型

有时提升一个模型的准确度很困难。如果你曾纠结于类似的问题，那我相信你会同意我的看法。你会尝试所有曾学习过的策略和算法，但模型正确率并没有改善。这时你会觉得无助和困顿，这也是 90% 的数据科学家开始放弃的时候。不过，这才是考验真本领的时候！这也是普通的数据科学家和大师级数据科学家的差距所在。

16 集成算法

前面介绍了一系列算法，每种算法都有不同的适用范围。在现实生活中，常常采用集体智慧来解决问题。那么在机器学习中，能否将多种机器学习算法组合在一起，使计算出来的结果更好呢？这就是集成算法的思想。集成算法是提高算法准确度的有效方法之一，本章就介绍如何通过 scikit-learn 来实现集成算法。本章将会介绍以下几种算法：

- 装袋（Bagging）算法。
- 提升（Boosting）算法。
- 投票（Voting）算法。

16.1 集成的方法

下面是三种流行的集成算法的方法。

- **装袋（Bagging）算法**：先将训练集分离成多个子集，然后通过各个子集训练多个模型。
- **提升（Boosting）算法**：训练多个模型并组成一个序列，序列中的每一个模型都会修正前一个模型的错误。

- **投票（Voting）算法**：训练多个模型，并采用样本统计来提高模型的准确度。

本章只简单地介绍一下相关的集成算法。在这里采用 Pima Indians 数据集，并用 10 折交叉验证来分离数据，再通过相应的评估矩阵来评估算法模型。

16.2 装袋算法

装袋算法是一种提高分类准确率的算法，通过给定组合投票的方式获得最优解。比如你生病了，去 *n* 个医院看了 *n* 个医生，每个医生都给你开了药方，最后哪个药方的出现次数多，就说明这个药方越有可能是最优解，这很好理解，这也是装袋算法的思想。下面将介绍三种装袋模型：

- 装袋决策树（Bagged Decision Trees）。
- 随机森林（Random Forest）。
- 极端随机树（Extra Trees）。

16.2.1 装袋决策树

装袋算法在数据具有很大的方差时非常有效，最常见的例子就是决策树的装袋算法。下面将在 scikit-learn 中通过 BaggingClassifier 实现分类与回归树算法。本例中创建了 100 棵决策树，代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
```

```
Y = array[:, 8]
num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)
cart = DecisionTreeClassifier()
num_tree = 100
model = BaggingClassifier(base_estimator=cart, n_estimators=num_tree,
random_state=seed)
result = cross_val_score(model, X, Y, cv=kfold)
print(result.mean())
```

与第 12 章的分类与回归树的结果（0.701708817498）比较，发现结果有了很大的提升。执行结果如下：

```
0.770745044429
```

16.2.2 随机森林

顾名思义，随机森林是用随机的方式建立一个森林，森林由很多的决策树组成，而且每一棵决策树之间是没有关联的。得到森林之后，当有一个新的输入样本进入的时候，就让森林中的每一棵决策树分别进行判断，看看这个样本应该属于哪一类，再看看哪一类被选择最多，就预测这个样本为哪一类。

在建立每一棵决策树的过程中，有两点需要注意：采样与完全分裂。首先是两个随机采样的过程，随机森林对输入的数据要进行行、列的采样。对于行采样采用有放回的方式，也就是在采样得到的样本集合中可能有重复的样本。假设输入样本为 N 个，那么采样的样本也为 N 个。这样在训练的时候，每一棵树的输入样本都不是全部的样本，就相对不容易出现过拟合。然后进行列采样，从 M 个 feature 中选出 m 个（$m << M$）。之后再对采样之后的数据使用完全分裂的方式建立决策树，这样决策树的某一个叶子节点要么是无法继续分裂的，要么所有样本都指向同一个分类。一般很多的决策树算法都有一个重要的步骤——剪枝，但是这里不这么做，因为之前的两个随机采样过程保证了随机性，所以不剪枝也不会出现过拟合。

这种算法得到的随机森林中的每一棵决策树都是很弱的，但是将它们组合起来就会

很厉害了。我觉得可以这样比喻随机森林算法：每一棵决策树就是一个精通某一个领域的专家，这样在随机森林中就有了很多个精通不同领域的专家，对于一个新的问题（新的输入数据），可以从不同的角度去看待它，最终由各个专家投票得到结果。

这种算法在 scikit-learn 中的实现类是 RandomForestClassifier。下面的例子是实现了 100 棵树的随机森林。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)
num_tree = 100
max_features = 3
model = RandomForestClassifier(n_estimators=num_tree, random_state=seed,
max_features=max_features)
result = cross_val_score(model, X, Y, cv=kfold)
print(result.mean())
```

执行结果如下：

```
0.77337662337
```

16.2.3 极端随机树

极端随机树是由 PierreGeurts 等人于 2006 年提出的，它与随机森林十分相似，都是由许多决策树构成。但它与随机森林有两个主要的区别：

（1）随机森林应用的是 Bagging 模型，而极端随机树是使用所有的训练样本得到每棵决策树，也就是每棵决策树应用的是相同的全部训练样本。

（2）随机森林是在一个随机子集内得到最优分叉特征属性，而极端随机树是完全随机地选择分叉特征属性，从而实现对决策树进行分叉的。

它在 scikit-learn 中的实现类是 ExtraTreesClassifier。下面的例子是实现了 100 棵树和 7 个随机特征的极端随机树。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import ExtraTreesClassifier
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)
num_tree = 100
max_features = 7
model = ExtraTreesClassifier(n_estimators=num_tree, random_state=seed,
max_features=max_features)
result = cross_val_score(model, X, Y, cv=kfold)
print(result.mean())
```

执行结果如下：

```
0.762987012987
```

16.3 提升算法

提升算法是一种用来提高弱分类算法准确度的方法，这种方法先构造一个预测函数系列，然后以一定的方式将它们组合成一个预测函数。提升算法也是一种提高任意给定学习算法准确度的方法，它是一种集成算法，主要通过对样本集的操作获得样本子集，然后用弱分类算法在样本子集上训练生成一系列的基分类器。它可以用来提高其他弱分类算法的识别率，也就是将其他的弱分类算法作为基分类算法放于提升框架中，通过提升框架对训练样本集的操作，得到不同的训练样本子集，再用该样本子集去训练生成基分类器。每得到一个样本集就用该基分类算法在该样本集上产生一个基分类器，这样在给定训练轮数 n 后，就可产生 n 个基分类器，然后提升算法将这 n 个基分类器进行加权融合，产生最后的结果分类器。在这 n 个基分类器中，每个分类器的识别率不一定很高，但它们联合后的结果有很高的识别率，这样便提高了弱分类算法的识别率。下面是两个非常常见的用于机器学习的提升算法：

- AdaBoost。
- 随机梯度提升（Stochastic Gradient Boosting）。

16.3.1 AdaBoost

AdaBoost 是一种迭代算法，其核心思想是针对同一个训练集训练不同的分类器（弱分类器），然后把这些弱分类器集合起来，构成一个更强的最终分类器（强分类器）。其算法本身是通过改变数据分布来实现的，它根据每次训练集中每个样本的分类是否正确，以及上次的总体分类的准确率，来确定每个样本的权值。它将修改过权值的新数据集送给下层分类器进行训练，再将每次训练得到的分类器融合起来，作为最后的决策分类器。使用 AdaBoost 分类器可以排除一些不必要的训练数据特征，并放在关键的训练数据上面。在 scikit-learn 中的实现类是 AdaBoostClassifier。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import AdaBoostClassifier
```

```
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)
num_tree = 30
model = AdaBoostClassifier(n_estimators=num_tree, random_state=seed)
result = cross_val_score(model, X, Y, cv=kfold)
print(result.mean())
```

执行结果如下：

```
0.76045796309
```

16.3.2 随机梯度提升

随机梯度提升法（GBM）基于的思想是：要找到某个函数的最大值，最好的办法就是沿着该函数的梯度方向探寻。梯度算子总是指向函数值增长最快的方向。由于梯度提升算法在每次更新数据集时都需要遍历整个数据集，计算复杂度较高，于是有了一个改进算法——随机梯度提升算法，该算法一次只用一个样本点来更新回归系数，极大地改善了算法的计算复杂度。在 scikit-learn 中的实现类是 GradientBoostingClassifier。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import GradientBoostingClassifier
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
```

```
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)
num_tree = 100
model = GradientBoostingClassifier(n_estimators=num_tree, random_state=seed)
result = cross_val_score(model, X, Y, cv=kfold)
print(result.mean())
```

执行结果如下：

```
0.766900205058
```

16.4 投票算法

投票算法（Voting）是一个非常简单的多个机器学习算法的集成算法。投票算法是通过创建两个或多个算法模型，利用投票算法将这些算法包装起来，计算各个子模型的平均预测状况。在实际的应用中，可以对每个子模型的预测结果增加权重，以提高算法的准确度。但是，在 scikit-learn 中不提供加权算法。下面通过一个例子来展示在 scikit-learn 中如何实现一个投票算法。在 scikit-learn 中的实现类是 VotingClassifier。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
```

```
X = array[:, 0:8]
Y = array[:, 8]
num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)
cart = DecisionTreeClassifier()
models = []
model_logistic = LogisticRegression()
models.append(('logistic', model_logistic))
model_cart = DecisionTreeClassifier()
models.append(('cart', model_cart))
model_svc = SVC()
models.append(('svm', model_svc))
ensemble_model = VotingClassifier(estimators=models)
result = cross_val_score(ensemble_model, X, Y, cv=kfold)
print(result.mean())
```

执行结果如下：

```
0.732997265892
```

16.5 总结

本章介绍了三种提高算法准确度的集成算法，下一章将会介绍另外一种提升算法准确度的方法——算法调参。

17 算法调参

机器学习的模型都是参数化的，可以通过调参来提高模型的准确度。模型有很多参数，如何找到最佳的参数组合，可以把它当作一个查询问题来处理，但是调整参数到何时为止呢？应该遵循偏差和方差协调的原则。本章将介绍在 scikit-learn 中设置机器学习模型最佳参数的方法。在本章将会介绍以下内容：

- 调整参数对机器学习算法的重要性。
- 如何使用网格搜索优化参数。
- 如何使用随机搜索优化参数。

17.1 机器学习算法调参

调整算法参数是采用机器学习解决问题的最后一个步骤，有时也被称为超参数优化。学会调参是进行机器学习项目的前提，但第一次遇到这些算法和模型时，肯定会被其大量的参数吓到。其实，参数可分为两种：一种是影响模型在训练集上的准确度或防止过拟合能力的参数；另一种是不影响这两者的参数。模型在样本总体上的准确度由其在训练集上的准确度及其防止过拟合的能力共同决定，所以在调参时主要针对第一种参数进

行调整，最终达到的效果是：模型在训练集上的准确度和防止过拟合能力的大和谐。下面将介绍两种自动寻找最优化参数的算法：

- 网格搜索优化参数。
- 随机搜索优化参数。

17.2 网格搜索优化参数

网格搜索优化参数是一种算法参数优化的方法。它是通过遍历已定义参数的列表，来评估算法的参数，从而找到最优参数。在 scikit-learn 中使用 GridSearchCV 来实现对参数的跟踪、调整与评估，从而找到最优参数。网格搜索优化参数适用于三四个（或更少）的超参数（当超参数的数量增加时，网格搜索的计算复杂度会呈现指数型增长，这时要换用随机搜索），由用户列出一个较小的超参数值域，这些超参数值域的笛卡尔集（排列组合）为一组组超参数。网格搜索算法使用每组超参数训练模型，并挑选验证集误差最小的超参数组合。下面的例子是展示如何使用 GridSearchCV 来调整脊回归（Ridge）的参数。GridSearchCV 使用字典对象来指定需要调参的参数，可以同时对一个或多个参数进行调参。代码如下：

```
from pandas import read_csv
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
# 算法实例化
model = Ridge()
# 设置要遍历的参数
param_grid = {'alpha': [1, 0.1, 0.01, 0.001, 0]}
```

```
# 通过网格搜索查询最优参数
grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid.fit(X, Y)
# 搜索结果
print('最高得分：%.3f' % grid.best_score_)
print('最优参数：%s' % grid.best_estimator_.alpha)
```

param_grid 是一个字典对象，以算法的参数名为 key，需要遍历的参数值列表为 value。在验证算法最优参数的网格搜索算法中，可以设定多个 key: value 对，同时查询多个参数的最优参数值。执行结果如下：

```
最高得分：0.280
最优参数：1
```

17.3 随机搜索优化参数

随机搜索优化参数是另一种对算法参数优化的方法。随机搜索优化参数通过固定次数的迭代，采用随机采样分布的方式搜索合适的参数。与网格搜索优化参数相比，随机搜索优化参数提供了一种更高效的解决方法（特别是在参数数量多的情况下），随机搜索优化参数为每个参数定义了一个分布函数，并在该空间中采样。在 scikit-learn 中通过 RandomizedSearchCV 类实现。下面的例子是通过 RandomizedSearchCV 对脊回归算法的参数进行 100 次迭代，并从中选择最优的参数。SciPy 中的 uniform 是一个均匀随机采样函数，默认生成 0 与 1 之间的随机采样数值。在这里利用 uniform 对参数进行随机采样。代码如下：

```
from pandas import read_csv
from sklearn.linear_model import Ridge
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
```

```
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
# 算法实例化
model = Ridge()
# 设置要遍历的参数
param_grid = {'alpha': uniform()}
# 通过网格搜索查询最优参数
grid = RandomizedSearchCV(estimator=model, param_distributions=param_grid,
n_iter=100, random_state=7)
grid.fit(X, Y)
# 搜索结果
print('最高得分：%.3f' % grid.best_score_)
print('最优参数：%s' % grid.best_estimator_.alpha)
```

执行结果如下：

```
最高得分：0.280
最优参数：0.977989511997
```

17.4 总结

调参是算法模型生成之前很重要的一步，本章介绍了两种选择最优参数的方法：网格搜索优化参数和随机搜索优化参数。如果算法的参数少于三个，推荐使用网格搜索优化参数；如果需要优化的参数超过三个，推荐使用随机搜索优化参数。至此，在模型生成之前的所有步骤都介绍完了。下一章将介绍如何生成模型，以及如何保存和载入已生成的模型。

第六部分

结果部署

结果部署是机器学习项目中的最后一步，也是最重要的步骤之一。选定算法之后，对算法训练生成模型，并部署到生产环境上，以便利用机器学习解决实际问题。模型生成之后，也需要定期对模型进行更新，使模型处于最新、最有效的状态，通常建议 3~6 个月更新一次模型。

18 持久化加载模型

找到一个能够生成高准确度模型的算法不是机器学习最后的步骤，在实际的项目中，需要将生成的模型序列化，并将其发布到生产环境。当有新数据出现时，需要反序列化已保存的模型，然后用其预测新的数据。本章将介绍在 Python 中如何序列化和反序列化 scikit-learn 的模型。本章内容将包括以下几个方面：

- 模型序列化和重用的重要性。
- 如何通过 pickle 来序列化和反序列化机器学习的模型。
- 如何通过 joblib 来序列化和反序列化机器学习的模型。

18.1 通过 pickle 序列化和反序列化机器学习的模型

pickle 是标准的 Python 序列化的方法，可以通过它来序列化机器学习算法生成的模型，并将其保存到文件中。当需要对新数据进行预测时，将保存在文件中的模型反序列化，并用其来预测新数据的结果。下面给出一个根据 Pima Indians 数据集训练逻辑回归算法生成的一个模型，并将其序列化到文件，然后反序列化这个模型的例子。在机器学习项目中，当模型训练需要花费大量的时间时，模型序列化是尤为重要的。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from pickle import dump
from pickle import load
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
test_size = 0.33
seed = 4
X_train, X_test, Y_traing, Y_test = train_test_split(X, Y, test_size=
test_size, random_state=seed)
# 训练模型
model = LogisticRegression()
model.fit(X_train, Y_traing)

# 保存模型
model_file = 'finalized_model.sav'
with open(model_file, 'wb') as model_f:
    # 模型序列化
    dump(model, model_f)

# 加载模型
with open(model_file, 'rb') as model_f:
    # 模型反序列化
    loaded_model = load(model_f)
    result = loaded_model.score(X_test, Y_test)
    print("算法评估结果：%.3f%%" % (result * 100))
```

执行结果如下：

```
算法评估结果：80.315%
```

18.2 通过 joblib 序列化和反序列化机器学习的模型

joblib 是 SciPy 生态环境的一部分，提供了通用的工具来序列化 Python 的对象和反序列化 Python 的对象。通过 joblib 序列化对象时会采用 NumPy 的格式保存数据，这对某些保存数据到模型中的算法非常有效，如 K 近邻算法。下面就是使用 joblib 进行序列化和反序列化的例子。代码如下：

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.externals.joblib import dump
from sklearn.externals.joblib import load
# 导入数据
filename = 'pima_data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
# 将数据分为输入数据和输出结果
array = data.values
X = array[:, 0:8]
Y = array[:, 8]
test_size = 0.33
seed = 4
X_train, X_test, Y_traing, Y_test = train_test_split(X, Y, test_size=
test_size, random_state=seed)
# 训练模型
model = LogisticRegression()
model.fit(X_train, Y_traing)

# 保存模型
model_file = 'finalized_model_joblib.sav'
with open(model_file, 'wb') as model_f:
    dump(model, model_f)

# 加载模型
with open(model_file, 'rb') as model_f:
    loaded_model = load(model_f)
```

```
result = loaded_model.score(X_test, Y_test)
print("算法评估结果：%.3f%%" % (result * 100))
```

执行结果如下：

```
算法评估结果：80.315%
```

18.3 生成模型的技巧

在生成机器学习模型时，需要考虑以下几个问题。

- **Python 的版本**：要记录下 Python 的版本，大部分情况下，在序列化模型和反序列化模型时，需要使用相同的 Python 版本。
- **类库版本**：同样需要记录所有的主要类库的版本，因为在序列化模型和反序列化模型时需要使用相同版本的类库，不仅需要 SciPy 和 scikit-learn 版本一致，其他的类库版本也需要一致。
- **手动序列化**：有时需要手动序列化算法参数，这样可以直接在 scikit-learn 中或其他的平台重现这个模型。我们通常会花费大量的时间在选择算法和参数调整上，将这个过程手动记录下来比仅序列化模型更有价值。

18.4 总结

本章介绍了通过 pickle 和 joblib 序列化和反序列化算法模型，至此所有的知识点就介绍结束了。在接下来的部分会将这些知识点整合到一起，完成一个机器学习的工程项目。

第七部分 项目实践

机器学习是一项经验技能，经验越多越好。在项目建立的过程中，实践是掌握机器学习的最佳手段。在实践过程中，通过实际操作加深对分类和回归问题的每一个步骤的理解，达到学习机器学习的目的。

19 预测模型项目模板

不能只通过阅读来掌握机器学习的技能，需要进行大量的练习。本章将介绍一个通用的机器学习的项目模板，创建这个模板总共有六个步骤。通过本章将学到：

- 端到端地预测（分类与回归）模型的项目结构。
- 如何将前面学到的内容引入到项目中。
- 如何通过这个项目模板来得到一个高准确度的模板。

机器学习是针对数据进行自动挖掘，找出数据的内在规律，并应用这个规律来预测新数据，如图 19-1 所示。

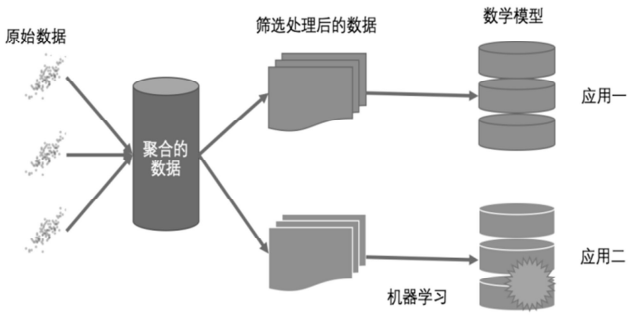


图 19-1

19.1 在项目中实践机器学习

端到端地解决机器学习的问题是非常重要的。可以学习机器学习的知识，可以实践机器学习的某个方面，但是只有针对某一个问题，从问题定义开始到模型部署为止，通过实践机器学习的各个方面，才能真正掌握并应用机器学习来解决实际问题。

在部署一个项目时，全程参与到项目中可以更加深入地思考如何使用模型，以及勇于尝试用机器学习解决问题的各个方面，而不仅仅是参与到自己感兴趣或擅长的方面。一个很好的实践机器学习项目的方法是，使用从 UCI 机器学习仓库（http://archive.ics.uci.edu/ml/datasets.html）获取的数据集开启一个机器学习项目。如果从一个数据集开始实践机器学习，应该如何将学到的所有技巧和方法整合到一起来处理机器学习的问题呢?

分类或回归模型的机器学习项目可以分成以下六个步骤：

（1）定义问题。

（2）理解数据。

（3）数据准备。

（4）评估算法。

（5）优化模型。

（6）结果部署。

有时这些步骤可能被合并或进一步分解，但通常是按上述六个步骤来开展机器学习项目的。为了符合 Python 的习惯，在下面的 Python 项目模板中，按照这六个步骤分解整个项目，在接下来的部分会明确各个步骤或子步骤中所要实现的功能。

19.2 机器学习项目的 Python 模板

下面会给出一个机器学习项目的 Python 模板。代码如下：

```
# Python 机器学习项目的模板

# 1. 定义问题
# a) 导入类库
# b) 导入数据集

# 2. 理解数据
# a) 描述性统计
# b) 数据可视化

# 3. 数据准备
# a) 数据清洗
# b) 特征选择
# c) 数据转换

# 4. 评估算法
# a) 分离数据集
# b) 定义模型评估标准
# c) 算法审查
# d) 算法比较

# 5. 优化模型
# a) 算法调参
# b) 集成算法

# 6. 结果部署
# a) 预测评估数据集
# b) 利用整个数据集生成模型
# c) 序列化模型
```

当有新的机器学习项目时，新建一个 Python 文件，并将这个模板粘贴进去，再按照前面章节介绍的方法将其填充到每一个步骤中。

19.3 各步骤的详细说明

接下来将详细介绍项目模板的各个步骤。

步骤 1：定义问题

主要是导入在机器学习项目中所需要的类库和数据集等，以便完成机器学习的项目，包括导入 Python 的类库、类和方法，以及导入数据。同时这也是所有的配置参数的配置模块。当数据集过大时，可以在这里对数据集进行瘦身处理，理想状态是可以在 1 分钟内，甚至是 30 秒内完成模型的建立或可视化数据集。

步骤 2：理解数据

这是加强对数据理解的步骤，包括通过描述性统计来分析数据和通过可视化来观察数据。在这一步需要花费时间多问几个问题，设定假设条件并调查分析一下，这对模型的建立会有很大的帮助。

步骤 3：数据准备

数据准备主要是预处理数据，以便让数据可以更好地展示问题，以及熟悉输入与输出结果的关系。包括：

- 通过删除重复数据、标记错误数值，甚至标记错误的输入数据来清洗数据。
- 特征选择，包括移除多余的特征属性和增加新的特征属性。
- 数据转化，对数据尺度进行调整，或者调整数据的分布，以便更好地展示问题。

要不断地重复这个步骤和下一个步骤，直到找到足够准确的算法生成模型。

步骤 4：评估算法

评估算法主要是为了寻找最佳的算法子集，包括：

- 分离出评估数据集，以便于验证模型。
- 定义模型评估标准，用来评估算法模型。
- 抽样审查线性算法和非线性算法。
- 比较算法的准确度。

在面对一个机器学习的问题的时候，需要花费大量的时间在评估算法和准备数据上，直到找到 3~5 种准确度足够的算法为止。

步骤 5：优化模型

当得到一个准确度足够的算法列表后，要从中找出最合适的算法，通常有两种方法可以提高算法的准确度：

- 对每一种算法进行调参，得到最佳结果。
- 使用集合算法来提高算法模型的准确度。

步骤 6：结果部署

一旦认为模型的准确度足够高，就可以将这个模型序列化，以便有新数据时使用该模型来预测数据。

- 通过验证数据集来验证被优化过的模型。
- 通过整个数据集来生成模型。
- 将模型序列化，以便于预测新数据。

做到这一步的时候，就可以将模型展示并发布给相关人员。当有新数据产生时，就可以采用这个模型来预测新数据。

19.4 使用模板的小技巧

快速执行一遍：首先要快速地在项目中将模板中的每一个步骤执行一遍，这样会加强对项目每一部分的理解并给如何改进带来灵感。

循环：整个流程不是线性的，而是循环进行的，要花费大量的时间来重复各个步骤，尤其是步骤 3 或步骤 4（或步骤 3～步骤 5），直到找到一个准确度足够的模型，或者达到预定的周期。

尝试每一个步骤：跳过某个步骤很简单，尤其是不熟悉、不擅长的步骤。坚持在这

个模板的每一个步骤中做些工作，即使这些工作不能提高算法的准确度，但也许在后面的操作就可以改进并提高算法的准确度。即使觉得这个步骤不适用，也不要跳过这个步骤，而是减少该步骤所做的贡献。

定向准确度：机器学习项目的目标是得到一个准确度足够高的模型。每一个步骤都要为实现这个目标做出贡献。要确保每次改变都会给结果带来正向的影响，或者对其他的步骤带来正向的影响。在整个项目的每个步骤中，准确度只能向变好的方向移动。

按需适用：可以按照项目的需要来修改步骤，尤其是对模板中的各个步骤非常熟悉之后。需要把握的原则是，每一次改进都以提高算法模型的准确度为前提。

19.5 总结

本章介绍了预测模型项目的模板，这个模板适用于分类或回归问题。接下来将介绍机器学习中的一个回归问题的项目，这个项目比前面介绍的鸢尾花的例子更加复杂，会利用到本章介绍的每个步骤。

20 回归项目实例

机器学习是一项经验技能，实践是掌握机器学习、提高利用机器学习解决问题的能力的有效方法之一。那么如何通过机器学习来解决问题呢？本章将通过一个实例来一步一步地介绍一个回归问题。本章主要介绍以下内容：

- 如何端到端地完成一个回归问题的模型。
- 如何通过数据转换提高模型的准确度。
- 如何通过调参提高模型的准确度。
- 如何通过集成算法提高模型的准确度。

20.1 定义问题

在这个项目中将分析研究波士顿房价（Boston House Price）数据集，这个数据集中的每一行数据都是对波士顿周边或城镇房价的描述。数据是 1978 年统计收集的。数据中包含以下 14 个特征和 506 条数据（UCI 机器学习仓库中的定义）。

- **CRIM**：城镇人均犯罪率。
- **ZN**：住宅用地所占比例。

- **INDUS**：城镇中非住宅用地所占比例。
- **CHAS**：CHAS 虚拟变量，用于回归分析。
- **NOX**：环保指数。
- **RM**：每栋住宅的房间数。
- **AGE**：1940 年以前建成的自住单位的比例。
- **DIS**：距离 5 个波士顿的就业中心的加权距离。
- **RAD**：距离高速公路的便利指数。
- **TAX**：每一万美元的不动产税率。
- **PRTATIO**：城镇中的教师学生比例。
- **B**：城镇中的黑人比例。
- **LSTAT**：地区中有多少房东属于低收入人群。
- **MEDV**：自住房屋房价中位数。

通过对这些特征属性的描述，我们可以发现输入的特征属性的度量单位是不统一的，也许需要对数据进行度量单位的调整。

20.2 导入数据

首先导入在项目中需要的类库。代码如下：

```
# 导入类库
import numpy as np
from numpy import arange
from matplotlib import pyplot
from pandas import read_csv
from pandas import  set_option
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
```

```
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import mean_squared_error
```

接下来导入数据集到 Python 中，这个数据集也可以从 UCI 机器学习仓库下载，在导入数据集时还设定了数据属性特征的名字。代码如下：

```
# 导入数据
filename = 'housing.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
        'RAD', 'TAX', 'PRTATIO', 'B', 'LSTAT', 'MEDV']
data = read_csv(filename, names=names, delim_whitespace=True)
```

在这里对每一个特征属性设定了一个名称，以便于在后面的程序中使用它们。因为 CSV 文件是使用空格键做分隔符的，因此读入 CSV 文件时指定分隔符为空格键（delim_whitespace=True）。

20.3 理解数据

对导入的数据进行分析，便于构建合适的模型。

首先看一下数据维度，例如数据集中有多少条记录、有多少个数据特征。代码如下：

```
# 数据维度
print(dataset.shape)
```

执行之后我们可以看到总共有 506 条记录和 14 个特征属性，这与 UCI 提供的信息一致。

```
(506, 14)
```

再查看各个特征属性的字段类型。代码如下：

```
# 特征属性的字段类型
print(dataset.dtypes)
```

可以看到所有的特征属性都是数字，而且大部分特征属性都是浮点数，也有一部分特征属性是整数类型的。执行结果如下：

```
CRIM      float64
ZN        float64
INDUS     float64
CHAS        int64
NOX       float64
RM        float64
AGE       float64
DIS       float64
RAD         int64
TAX       float64
PRTATIO    float64
B         float64
LSTAT     float64
MEDV      float64
dtype: object
```

接下来对数据进行一次简单的查看，在这里我们查看一下最开始的 30 条记录。代码如下：

```
# 查看最开始的 30 条记录
set_option('display.line_width', 120)
print(dataset.head(30))
```

这里指定输出的宽度为 120 个字符，以确保将所有特征属性值显示在一行内。而且这些数据不是用相同的单位存储的，进行后面的操作时，也许需要将数据整理为相同的度量单位。执行结果如图 20-1 所示。

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PRTATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2
5	0.02985	0.0	2.18	0	0.458	6.430	58.7	6.0622	3	222.0	18.7	394.12	5.21	28.7
6	0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311.0	15.2	395.60	12.43	22.9
7	0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311.0	15.2	396.90	19.15	27.1
8	0.21124	12.5	7.87	0	0.524	5.631	100.0	6.0821	5	311.0	15.2	386.63	29.93	16.5
9	0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311.0	15.2	386.71	17.10	18.9
10	0.22489	12.5	7.87	0	0.524	6.377	94.3	6.3467	5	311.0	15.2	392.52	20.45	15.0
11	0.11747	12.5	7.87	0	0.524	6.009	82.9	6.2267	5	311.0	15.2	396.90	13.27	18.9
12	0.09378	12.5	7.87	0	0.524	5.889	39.0	5.4509	5	311.0	15.2	390.50	15.71	21.7
13	0.62976	0.0	8.14	0	0.538	5.949	61.8	4.7075	4	307.0	21.0	396.90	8.26	20.4
14	0.63796	0.0	8.14	0	0.538	6.096	84.5	4.4619	4	307.0	21.0	380.02	10.26	18.2
15	0.62739	0.0	8.14	0	0.538	5.834	56.5	4.4986	4	307.0	21.0	395.62	8.47	19.9
16	1.05393	0.0	8.14	0	0.538	5.935	29.3	4.4986	4	307.0	21.0	386.85	6.58	23.1
17	0.78420	0.0	8.14	0	0.538	5.990	81.7	4.2579	4	307.0	21.0	386.75	14.67	17.5
18	0.80271	0.0	8.14	0	0.538	5.456	36.6	3.7965	4	307.0	21.0	288.99	11.69	20.2
19	0.72580	0.0	8.14	0	0.538	5.727	69.5	3.7965	4	307.0	21.0	390.95	11.28	18.2
20	1.25179	0.0	8.14	0	0.538	5.570	98.1	3.7979	4	307.0	21.0	376.57	21.02	13.6
21	0.85204	0.0	8.14	0	0.538	5.965	89.2	4.0123	4	307.0	21.0	392.53	13.83	19.6
22	1.23247	0.0	8.14	0	0.538	6.142	91.7	3.9769	4	307.0	21.0	396.90	18.72	15.2
23	0.98843	0.0	8.14	0	0.538	5.813	100.0	4.0952	4	307.0	21.0	394.54	19.88	14.5
24	0.75026	0.0	8.14	0	0.538	5.924	94.1	4.3996	4	307.0	21.0	394.33	16.30	15.6
25	0.84054	0.0	8.14	0	0.538	5.599	85.7	4.4546	4	307.0	21.0	303.42	16.51	13.9
26	0.67191	0.0	8.14	0	0.538	5.813	90.3	4.6820	4	307.0	21.0	376.88	14.81	16.6
27	0.95577	0.0	8.14	0	0.538	6.047	88.8	4.4534	4	307.0	21.0	306.38	17.28	14.8
28	0.77299	0.0	8.14	0	0.538	6.495	94.4	4.4547	4	307.0	21.0	387.94	12.80	18.4
29	1.00245	0.0	8.14	0	0.538	6.674	87.3	4.2390	4	307.0	21.0	380.23	11.98	21.0

图 20-1

接下来看一下数据的描述性统计信息。代码如下：

```
# 描述性统计信息
set_option('precision', 1)
print(dataset.describe())
```

在描述性统计信息中包含数据的最大值、最小值、中位值、四分位值等，分析这些数据能够加深对数据分布、数据结构等的理解。结果如图 20-2 所示。

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PRTATIO	B	LSTAT	MEDV
count	5.1e+02	506.0	506.0	5.1e+02	506.0	506.0	506.0	506.0	506.0	506.0	506.0	506.0	506.0	506.0
mean	3.6e+00	11.4	11.1	6.9e−02	0.6	6.3	68.6	3.8	9.5	408.2	18.5	356.7	12.7	22.5
std	8.6e+00	23.3	6.9	2.5e−01	0.1	0.7	28.1	2.1	8.7	168.5	2.2	91.3	7.1	9.2
min	6.3e−03	0.0	0.5	0.0e+00	0.4	3.6	2.9	1.1	1.0	187.0	12.6	0.3	1.7	5.0
25%	8.2e−02	0.0	5.2	0.0e+00	0.4	5.9	45.0	2.1	4.0	279.0	17.4	375.4	6.9	17.0
50%	2.6e−01	0.0	9.7	0.0e+00	0.5	6.2	77.5	3.2	5.0	330.0	19.1	391.4	11.4	21.2
75%	3.7e+00	12.5	18.1	0.0e+00	0.6	6.6	94.1	5.2	24.0	666.0	20.2	396.2	17.0	25.0
max	8.9e+01	100.0	27.7	1.0e+00	0.9	8.8	100.0	12.1	24.0	711.0	22.0	396.9	38.0	50.0

图 20-2

接下来看一下数据特征之间的两两关联关系，这里查看数据的皮尔逊相关系数。代码如下：

```
# 关联关系
set_option('precision', 2)
print(dataset.corr(method='pearson'))
```

执行结果如图 20-3 所示。

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PRTATIO	B	LSTAT	MEDV
CRIM	1.00	-0.20	0.41	-5.59e-02	0.42	-0.22	0.35	-0.38	6.26e-01	0.58	0.29	-0.39	0.46	-0.39
ZN	-0.20	1.00	-0.53	-4.27e-02	-0.52	0.31	-0.57	0.66	-3.12e-01	-0.31	-0.39	0.18	-0.41	0.36
INDUS	0.41	-0.53	1.00	6.29e-02	0.76	-0.39	0.64	-0.71	5.95e-01	0.72	0.38	-0.36	0.60	-0.48
CHAS	-0.06	-0.04	0.06	1.00e+00	0.09	0.09	0.09	-0.10	-7.37e-03	-0.04	-0.12	0.05	-0.05	0.18
NOX	0.42	-0.52	0.76	9.12e-02	1.00	-0.30	0.73	-0.77	6.11e-01	0.67	0.19	-0.38	0.59	-0.43
RM	-0.22	0.31	-0.39	9.13e-02	-0.30	1.00	-0.24	0.21	-2.10e-01	-0.29	-0.36	0.13	-0.61	0.70
AGE	0.35	-0.57	0.64	8.65e-02	0.73	-0.24	1.00	-0.75	4.56e-01	0.51	0.26	-0.27	0.60	-0.38
DIS	-0.38	0.66	-0.71	-9.92e-02	-0.77	0.21	-0.75	1.00	-4.95e-01	-0.53	-0.23	0.29	-0.50	0.25
RAD	0.63	-0.31	0.60	-7.37e-03	0.61	-0.21	0.46	-0.49	1.00e+00	0.91	0.46	-0.44	0.49	-0.38
TAX	0.58	-0.31	0.72	-3.56e-02	0.67	-0.29	0.51	-0.53	9.10e-01	1.00	0.46	-0.44	0.54	-0.47
PRTATIO	0.29	-0.39	0.38	-1.22e-01	0.19	-0.36	0.26	-0.23	4.65e-01	0.46	1.00	-0.18	0.37	-0.51
B	-0.39	0.18	-0.36	4.88e-02	-0.38	0.13	-0.27	0.29	-4.44e-01	-0.44	-0.18	1.00	-0.37	0.33
LSTAT	0.46	-0.41	0.60	-5.39e-02	0.59	-0.61	0.60	-0.50	4.89e-01	0.54	0.37	-0.37	1.00	-0.74
MEDV	-0.39	0.36	-0.48	1.75e-01	-0.43	0.70	-0.38	0.25	-3.82e-01	-0.47	-0.51	0.33	-0.74	1.00

图 20-3

通过上面的结果可以看到，有些特征属性之间具有强关联关系（>0.7 或<-0.7），如：

- NOX 与 INDUS 之间的皮尔逊相关系数是 0.76。
- DIS 与 INDUS 之间的皮尔逊相关系数是-0.71。
- TAX 与 INDUS 之间的皮尔逊相关系数是 0.72。
- AGE 与 NOX 之间的皮尔逊相关系数是 0.73。
- DIS 与 NOX 之间的皮尔逊相关系数是-0.77。

20.4 数据可视化

20.4.1 单一特征图表

首先查看每一个数据特征单独的分布图，多查看几种不同的图表有助于发现更好的方法。我们可以通过查看各个数据特征的直方图，来感受一下数据的分布情况。代码如下：

```
 # 直方图
dataset.hist(sharex=False, sharey=False, xlabelsize=1, ylabelsize=1)
pyplot.show()
```

执行结果如图 20-4 所示，从图中可以看到有些数据呈指数分布，如 CRIM、ZN、AGE 和 B；有些数据特征呈双峰分布，如 RAD 和 TAX。

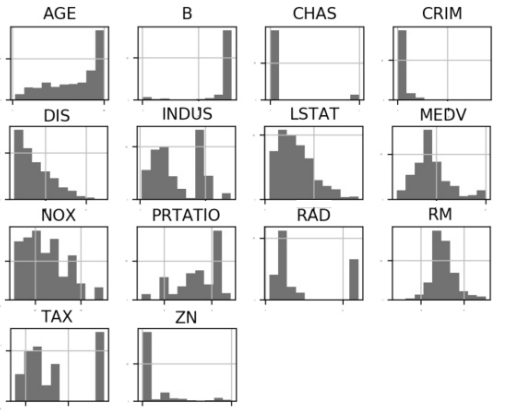


图 20-4

通过密度图可以展示这些数据的特征属性，密度图比直方图更加平滑地展示了这些数据特征。代码如下：

```
 # 密度图
dataset.plot(kind='density', subplots=True, layout=(4,4), sharex=False,
fontsize=1)
pyplot.show()
```

在密度图中，指定 layout=(4, 4)，这说明要画一个四行四列的图形。执行结果如图 20-5 所示。

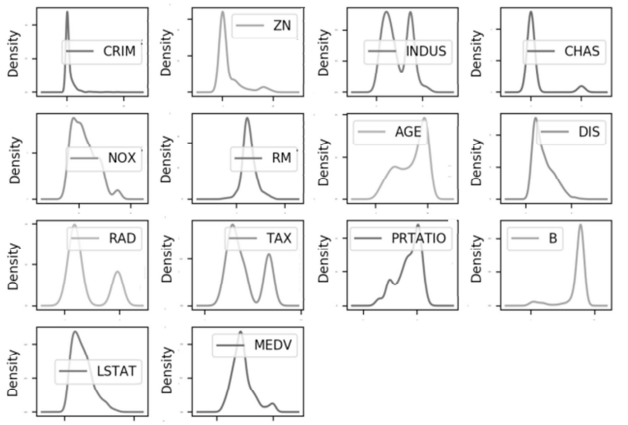


图 20-5

通过箱线图可以查看每一个数据特征的状况，也可以很方便地看出数据分布的偏态程度。代码如下：

```
# 箱线图
dataset.plot(kind='box', subplots=True, layout=(4,4), sharex=False,
sharey=False, fontsize=8)
pyplot.show()
```

执行结果如图 20-6 所示。

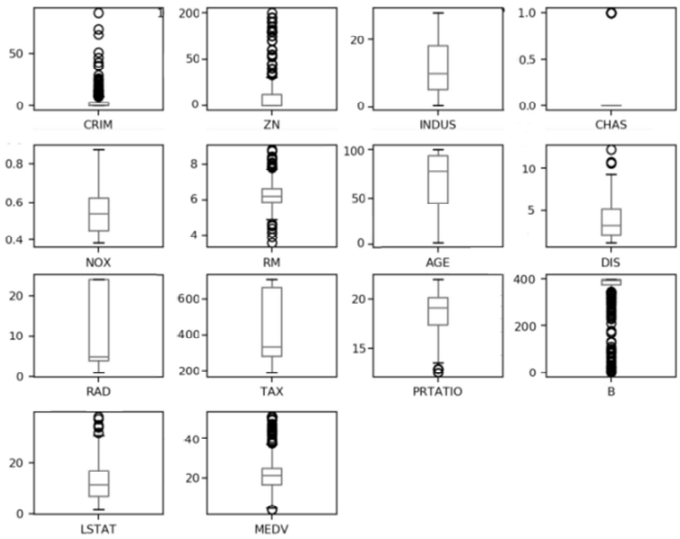


图 20-6

20.4.2 多重数据图表

接下来利用多重数据图表来查看不同数据特征之间的相互影响关系。首先看一下散点矩阵图。代码如下：

```
# 散点矩阵图
scatter_matrix(dataset)
pyplot.show()
```

通过散点矩阵图可以看到，虽然有些数据特征之间的关联关系很强，但是这些数据

分布结构也很好。即使不是线性分布结构，也是可以很方便进行预测的分布结构，执行结果如图 20-7 所示。

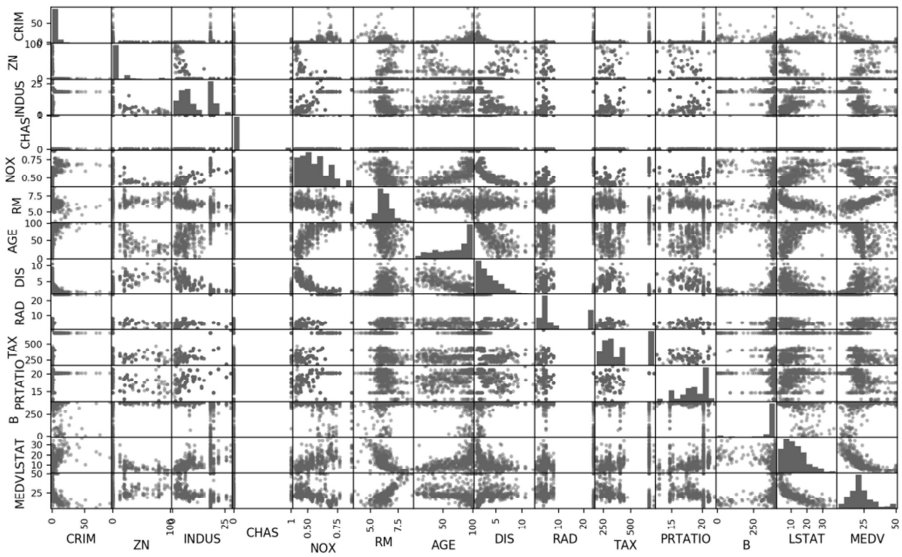


图 20-7

再看一下数据相互影响的相关矩阵图。代码如下：

```
 # 相关矩阵图
fig = pyplot.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(dataset.corr(), vmin=-1, vmax=1, interpolation='none')
fig.colorbar(cax)
ticks = np.arange(0, 14, 1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(names)
ax.set_yticklabels(names)
pyplot.show()
```

执行结果如图 20-8 所示，根据图例可以看到，数据特征属性之间的两两相关性，有些属性之间是强相关的，建议在后续的处理中移除这些特征属性，以提高算法的准确度。

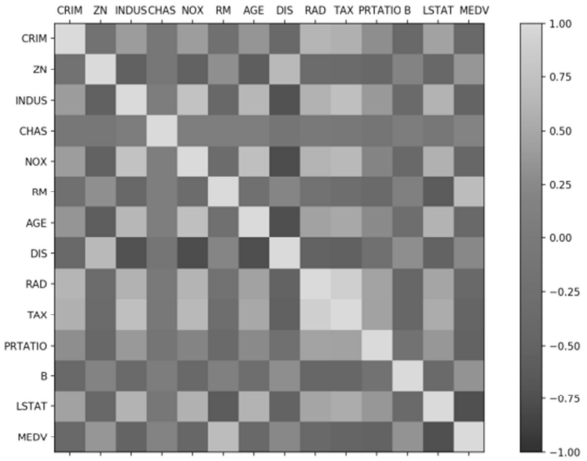


图 20-8

20.4.3 思路总结

通过数据的相关性和数据的分布等发现，数据集中的数据结构比较复杂，需要考虑对数据进行转换，以提高模型的准确度。可以尝试从以下几个方面对数据进行处理：

- 通过特征选择来减少大部分相关性高的特征。
- 通过标准化数据来降低不同数据度量单位带来的影响。
- 通过正态化数据来降低不同的数据分布结构，以提高算法的准确度。

可以进一步查看数据的可能性分级（离散化），它可以帮助提高决策树算法的准确度。

20.5 分离评估数据集

分离出一个评估数据集是一个很好的主意，这样可以确保分离出的数据集与训练模型的数据集完全隔离，有助于最终判断和报告模型的准确度。在进行到项目的最后一步处理时，会使用这个评估数据集来确认模型的准确度。这里分离出 20%的数据作为评估数据集，80%的数据作为训练数据集。代码如下：

```
# 分离数据集
array = dataset.values
X = array[:, 0:13]
Y = array[:, 13]
validation_size = 0.2
seed = 7
X_train, X_validation, Y_train, Y_validation = train_test_split(X,
Y,test_size=validation_size, random_state=seed)
```

20.6 评估算法

20.6.1 评估算法——原始数据

分析完数据不能立刻选择出哪个算法对需要解决的问题最有效。我们直观上认为，由于部分数据的线性分布，线性回归算法和弹性网络回归算法对解决问题可能比较有效。另外，由于数据的离散化，通过决策树算法或支持向量机算法也许可以生成高准确度的模型。到这里，依然不清楚哪个算法会生成准确度最高的模型，因此需要设计一个评估框架来选择合适的算法。我们采用 10 折交叉验证来分离数据，通过均方误差来比较算法的准确度。均方误差越趋近于 0，算法准确度越高。代码如下：

```
# 评估算法 —— 评估标准
num_folds = 10
seed = 7
scoring = 'neg_mean_squared_error'
```

对原始数据不做任何处理，对算法进行一个评估，形成一个算法的评估基准。这个基准值是对后续算法改善优劣比较的基准值。我们选择三个线性算法和三个非线性算法来进行比较。

线性算法：线性回归（LR）、套索回归（LASSO）和弹性网络回归（EN）。

非线性算法：分类与回归树（CART）、支持向量机（SVM）和 K 近邻算法（KNN）。

算法模型初始化的代码如下：

```
# 评估算法 - baseline
```

```
models = {}
models['LR'] = LinearRegression()
models['LASSO'] = Lasso()
models['EN'] = ElasticNet()
models['KNN']  = KNeighborsRegressor()
models['CART'] = DecisionTreeRegressor()
models['SVM'] = SVR()
```

对所有的算法使用默认参数，并比较算法的准确度，此处比较的是均方误差的均值和标准方差。代码如下：

```
 # 评估算法
results = []
for key in models:
    kfold = KFold(n_splits=num_folds, random_state=seed)
    cv_result = cross_val_score(models[key], X_train, Y_train, cv=kfold,
scoring=scoring)
    results.append(cv_result)
    print('%s: %f (%f)' % (key, cv_result.mean(), cv_result.std()))
```

从执行结果来看，线性回归（LR）具有最优的 MSE，接下来是分类与回归树（CART）算法。执行结果如下：

```
LR: -21.379856 (9.414264)
LASSO: -26.423561 (11.651110)
EN: -27.502259 (12.305022)
KNN: -41.896488 (13.901688)
CART: -26.608476 (12.250800)
SVM: -85.518342 (31.994798)
```

再查看所有的 10 折交叉分离验证的结果。代码如下：

```
 #评估算法——箱线图
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(models.keys())
pyplot.show()
```

执行结果如图 20-9 所示，从图中可以看到，线性算法的分布比较类似，并且 K 近邻算法的结果分布非常紧凑。

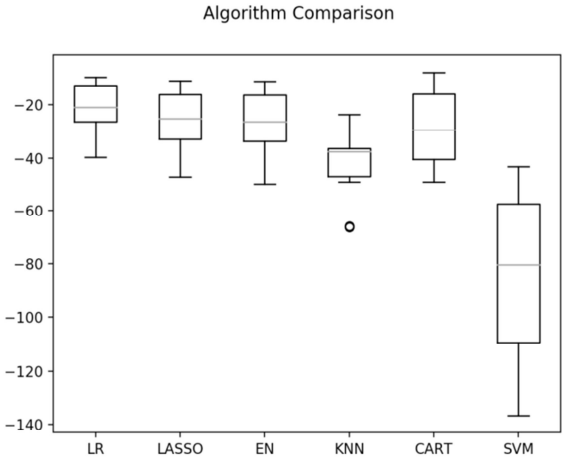


图 20-9

不同的数据度量单位，也许是 K 近邻算法和支持向量机算法表现不佳的主要原因。下面将对数据进行正态化处理，再次比较算法的结果。

20.6.2 评估算法——正态化数据

在这里猜测也许因为原始数据中不同特征属性的度量单位不一样，导致有的算法的结果不是很好。接下来通过对数据进行正态化，再次评估这些算法。在这里对训练数据集进行数据转换处理，将所有的数据特征值转化成“0”为中位值、标准差为“1”的数据。对数据正态化时，为了防止数据泄露，采用 Pipeline 来正态化数据和对模型进行评估。为了与前面的结果进行比较，此处采用相同的评估框架来评估算法模型。代码如下：

```
 # 评估算法——正态化数据
pipelines = {}
pipelines['ScalerLR'] = Pipeline([('Scaler', StandardScaler()), ('LR',
LinearRegression())])
pipelines['ScalerLASSO'] = Pipeline([('Scaler', StandardScaler()), ('LASSO',
Lasso())])
```

```
pipelines['ScalerEN'] = Pipeline([('Scaler',
StandardScaler()), ('EN', ElasticNet())])
pipelines['ScalerKNN'] = Pipeline([('Scaler',
StandardScaler()), ('KNN', KNeighborsRegressor())])
pipelines['ScalerCART'] = Pipeline([('Scaler',
StandardScaler()), ('CART', DecisionTreeRegressor())])
pipelines['ScalerSVM'] = Pipeline([('Scaler',
StandardScaler()), ('SVM', SVR())])
results = []
for key in pipelines:
    kfold = KFold(n_splits=num_folds, random_state=seed)
    cv_result = cross_val_score(pipelines[key], X_train, Y_train, cv=kfold,
scoring=scoring)
    results.append(cv_result)
    print('%s: %f (%f)' % (key, cv_result.mean(), cv_result.std()))
```

执行后发现 K 近邻算法具有最优的 MSE。执行结果如下：

```
ScalerLR: -21.379856 (9.414264)
ScalerLASSO: -26.607314 (8.978761)
ScalerEN: -27.932372 (10.587490)
ScalerKNN: -20.107620 (12.376949)
ScalerCART: -26.978716 (12.164366)
ScalerSVM: -29.633086 (17.009186)
```

接下来看一下所有的 10 折交叉分离验证的结果。代码如下：

```
#评估算法——箱线图
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(models.keys())
pyplot.show()
```

执行结果，生成的箱线图如图 20-10 所示，可以看到 K 近邻算法具有最优的 MSE 和最紧凑的数据分布。

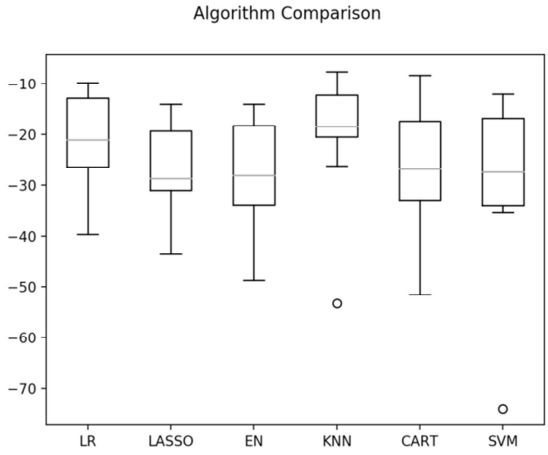


图 20-10

20.7 调参改善算法

目前来看，K 近邻算法对做过数据转换的数据集有很好的结果，但是是否可以进一步对结果做一些优化呢？K 近邻算法的默认参数近邻个数（n_neighbors）是 5，下面通过网格搜索算法来优化参数。代码如下：

```
 # 调参改善算法——KNN
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
param_grid = {'n_neighbors': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21]}
model = KNeighborsRegressor()
kfold = KFold(n_splits=num_folds, random_state=seed)
grid = GridSearchCV(estimator=model,
param_grid=param_grid, scoring=scoring, cv=kfold)
grid_result = grid.fit(X=rescaledX, y=Y_train)

print('最优：%s 使用%s' % (grid_result.best_score_, grid_result.best_params_))
cv_results =
zip(grid_result.cv_results_['mean_test_score'],

grid_result.cv_results_['std_test_score'],
```

```
                grid_result.cv_results_['params'])
for mean, std, param in cv_results:
    print('%f (%f) with %r' % (mean, std, param))
```

最优结果——K 近邻算法的默认参数近邻个数（n_neighbors）是 3。执行结果如下：

```
最优：-18.1721369637 使用{'n_neighbors': 3}
-20.208663 (15.029652) with {'n_neighbors': 1}
-18.172137 (12.950570) with {'n_neighbors': 3}
-20.131163 (12.203697) with {'n_neighbors': 5}
-20.575845 (12.345886) with {'n_neighbors': 7}
-20.368264 (11.621738) with {'n_neighbors': 9}
-21.009204 (11.610012) with {'n_neighbors': 11}
-21.151809 (11.943318) with {'n_neighbors': 13}
-21.557400 (11.536339) with {'n_neighbors': 15}
-22.789938 (11.566861) with {'n_neighbors': 17}
-23.871873 (11.340389) with {'n_neighbors': 19}
-24.361362 (11.914786) with {'n_neighbors': 21}
```

20.8 集成算法

除调参之外，提高模型准确度的方法是使用集成算法。下面会对表现比较好的线性回归、K 近邻、分类与回归树算法进行集成，来看看算法能否提高。

装袋算法：随机森林（RF）和极端随机树（ET）。

提升算法：AdaBoost（AB）和随机梯度上升（GBM）。

依然采用和前面同样的评估框架和正态化之后的数据来分析相关的算法。代码如下：

```
# 集成算法
ensembles = {}
ensembles['ScaledAB'] = Pipeline([('Scaler',
StandardScaler()), ('AB', AdaBoostRegressor())])
ensembles['ScaledAB-KNN'] = Pipeline([('Scaler',
StandardScaler()), ('ABKNN', AdaBoostRegressor
(base_estimator= KNeighborsRegressor(n_neighbors=3)))])
ensembles['ScaledAB-LR'] = Pipeline([('Scaler',
```

```
StandardScaler()), ('ABLR',
AdaBoostRegressor(LinearRegression()))])
ensembles['ScaledRFR'] = Pipeline([('Scaler',
StandardScaler()), ('RFR', RandomForestRegressor())])
ensembles['ScaledETR'] = Pipeline([('Scaler',
StandardScaler()), ('ETR', ExtraTreesRegressor())])
ensembles['ScaledGBR'] = Pipeline([('Scaler',
StandardScaler()), ('RBR', GradientBoostingRegressor())])

results = []
for key in ensembles:
    kfold = KFold(n_splits=num_folds, random_state=seed)
    cv_result = cross_val_score(ensembles[key], X_train, Y_train, cv=kfold,
scoring=scoring)
    results.append(cv_result)
    print('%s: %f (%f)' % (key, cv_result.mean(), cv_result.std()))
```

与前面的线性算法和非线性算法相比，这次的准确度都有了较大的提高。执行结果如下：

```
ScaledAB: -15.244803 (6.272186)
ScaledAB-KNN: -15.794844 (10.565933)
ScaledAB-LR: -24.108881 (10.165026)
ScaledRFR: -13.279674 (6.724465)
ScaledETR: -10.464980 (5.476443)
ScaledGBR: -10.256544 (4.605660)
```

接下来通过箱线图看一下集成算法在 10 折交叉验证中均方误差的分布状况。代码如下：

```
# 集成算法——箱线图
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(ensembles.keys())
pyplot.show()
```

执行结果如图 20-11 所示，随机梯度上升算法和极端随机树算法具有较高的中位值和分布状况。

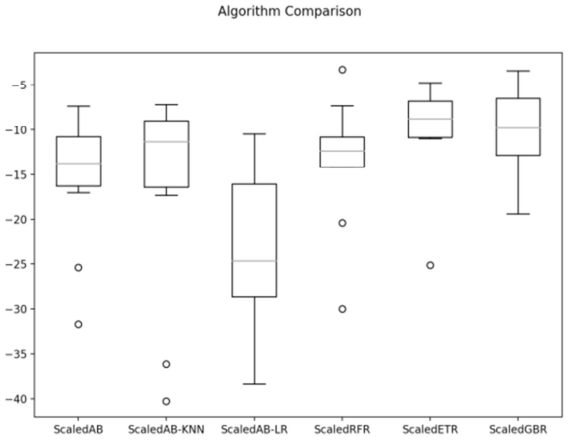


图 20-11

20.9 集成算法调参

集成算法都有一个参数 n_estimators，这是一个很好的可以用来调整的参数。对于集成参数来说，n_estimators 会带来更准确的结果，当然这也有一定的限度。下面对随机梯度上升（GBM）和极端随机树（ET）算法进行调参，再次比较这两个算法模型的准确度，来确定最终的算法模型。代码如下：

```
# 集成算法 GBM——调参
caler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
param_grid = {'n_estimators': [10, 50, 100, 200, 300, 400, 500, 600, 700, 800,
900]}
model = GradientBoostingRegressor()
kfold = KFold(n_splits=num_folds, random_state=seed)
grid = GridSearchCV(estimator=model,
param_grid=param_grid, scoring=scoring, cv=kfold)
grid_result = grid.fit(X=rescaledX, y=Y_train)
```

```
print('最优：%s 使用%s' % (grid_result.best_score_,
grid_result.best_params_))

# 集成算法 ET——调参
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
param_grid = {'n_estimators': [5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]}
model = ExtraTreesRegressor()
kfold = KFold(n_splits=num_folds, random_state=seed)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring,
cv=kfold)
grid_result = grid.fit(X=rescaledX, y=Y_train)
print('最优：%s 使用%s' % (grid_result.best_score_, grid_result.best_params_))
```

对于随机梯度上升（GBM）算法来说，最优的 n_estimators 是 500；对于极端随机树（ET）算法来说，最优的 n_estimators 是 80。执行结果，极端随机树（ET）算法略优于随机梯度上升（GBM）算法，因此采用极端随机树（ET）算法来训练最终的模型。执行结果如下：

```
最优：-9.3078229754 使用{'n_estimators': 500}
最优：-8.99113433246 使用{'n_estimators': 80}
```

也许需要执行多次这个过程才能找到最优参数。这里有一个技巧，当最优参数是 param_grid 的边界值时，有必要调整 param_grid 进行下一次调参。

20.10 确定最终模型

我们已经确定了使用极端随机树（ET）算法来生成模型，下面就对该算法进行训练和生成模型，并计算模型的准确度。代码如下：

```
#训练模型
caler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
gbr = ExtraTreesRegressor(n_estimators=80)
gbr.fit(X=rescaledX, y=Y_train)
```

再通过评估数据集来评估算法的准确度。

```
# 评估算法模型
rescaledX_validation = scaler.transform(X_validation)
predictions = gbr.predict(rescaledX_validation)
print(mean_squared_error(Y_validation, predictions))
```

执行结果如下：

```
14.077038511
```

20.11 总结

本项目实例从问题定义开始，直到最后的模型生成为止，完成了一个完整的机器学习项目。通过这个项目，理解了上一章中介绍的机器学习项目的模板，以及整个机器学习模型建立的流程。接下来会介绍一个机器学习的二分类问题，以进一步加深对这个模板的理解。

21 二分类实例

上一章通过一个回归分析的例子，完成了一个机器学习项目的实践。接下来会通过一个实例来介绍分类问题项目的流程。

- 如何端到端地完成一个分类问题的模型。
- 如何通过数据转换提高模型的准确度。
- 如何通过调参提高模型的准确度。
- 如何通过集成算法提高模型的准确度。

21.1 问题定义

在这个项目中将采用声呐、矿山和岩石数据集（http://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+%28Sonar%2C+Mines+vs.+Rocks%29），通过声呐返回的信息判断物质是金属还是岩石。这个数据集共有 208 条记录，每条数据记录了 60 种不同的声呐探测的数据和一个分类结果，若是岩石则标记为 R，若是金属则标记为 M。

21.2 导入数据

从 UCI 下载数据并保存在本地，与之前的实例一样，使用 Pandas 的 read_csv 来导入数据。在导入数据之前，先导入所有需要的类库。代码如下：

```
# 导入类库
import numpy as np
from matplotlib import pyplot
from pandas import read_csv
from pandas.plotting import scatter_matrix
from pandas import set_option
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier

# 导入数据
filename = 'sonar.all-data.csv'
dataset = read_csv(filename, header=None)
```

因为每一条记录都是 60 种不同的声呐探测结果，没有办法为它们命名合适的名字，所以这次导入数据时没有指定特征属性名称。数据导入完毕之后，可以查看数据，理解并分析数据。

21.3 分析数据

21.3.1 描述性统计

先确认一下数据的维度，例如记录的条数和数据特征属性的个数。代码如下：

```
# 数据维度
print(dataset.shape)
```

执行结果显示数据有 208 条记录和 61 个数据特征属性（包含 60 次声呐探测的数据和一个分类结果），这和 UCI 上对数据的描述一致。结果如下：

```
(208, 61)
```

接下来看一下各个数据特征的数据类型。代码如下：

```
# 查看数据类型
set_option('display.max_rows', 500)
print(dataset.dtypes)
```

从执行结果可以看到，所有的特征属性的数据类型都是数字。结果如下：

```
0    float64
1    float64
2    float64
3    float64
4    float64
5    float64
6    float64
7    float64
8    float64
9    float64
10   float64
11   float64
12   float64
13   float64
14   float64
15   float64
16   float64
17   float64
```

```
18   float64
19   float64
20   float64
21   float64
22   float64
23   float64
24   float64
25   float64
26   float64
27   float64
28   float64
29   float64
30   float64
31   float64
32   float64
33   float64
34   float64
35   float64
36   float64
37   float64
38   float64
39   float64
40   float64
41   float64
42   float64
43   float64
44   float64
45   float64
46   float64
47   float64
48   float64
49   float64
50   float64
51   float64
52   float64
53   float64
54   float64
55   float64
56   float64
```

```
57    float64
58    float64
59    float64
60     object
dtype: object
```

再查看最开始的 20 条记录。代码如下：

```
# 查看最开始的 20 条记录
set_option('display.width', 100)
print(dataset.head(20))
```

执行结果如下：

```
        0       1       2       3       4       5       6       7       8
9  ...     51  \
 0   0.0200  0.0371  0.0428  0.0207  0.0954  0.0986  0.1539  0.1601  0.3109
0.2111 ...  0.0027
 1   0.0453  0.0523  0.0843  0.0689  0.1183  0.2583  0.2156  0.3481  0.3337
0.2872 ...  0.0084
 2   0.0262  0.0582  0.1099  0.1083  0.0974  0.2280  0.2431  0.3771  0.5598
0.6194 ...  0.0232
 3   0.0100  0.0171  0.0623  0.0205  0.0205  0.0368  0.1098  0.1276  0.0598
0.1264 ...  0.0121
 4   0.0762  0.0666  0.0481  0.0394  0.0590  0.0649  0.1209  0.2467  0.3564
0.4459 ...  0.0031
 5   0.0286  0.0453  0.0277  0.0174  0.0384  0.0990  0.1201  0.1833  0.2105
0.3039 ...  0.0045
 6   0.0317  0.0956  0.1321  0.1408  0.1674  0.1710  0.0731  0.1401  0.2083
0.3513 ...  0.0201
 7   0.0519  0.0548  0.0842  0.0319  0.1158  0.0922  0.1027  0.0613  0.1465
0.2838 ...  0.0081
 8   0.0223  0.0375  0.0484  0.0475  0.0647  0.0591  0.0753  0.0098  0.0684
0.1487 ...  0.0145
 9   0.0164  0.0173  0.0347  0.0070  0.0187  0.0671  0.1056  0.0697  0.0962
0.0251 ...  0.0090
 10  0.0039  0.0063  0.0152  0.0336  0.0310  0.0284  0.0396  0.0272  0.0323
0.0452 ...  0.0062
 11  0.0123  0.0309  0.0169  0.0313  0.0358  0.0102  0.0182  0.0579  0.1122
0.0835 ...  0.0133
```

```
 12  0.0079  0.0086  0.0055  0.0250  0.0344  0.0546  0.0528  0.0958  0.1009
0.1240 ...  0.0176
 13  0.0090  0.0062  0.0253  0.0489  0.1197  0.1589  0.1392  0.0987  0.0955
0.1895 ...  0.0059
 14  0.0124  0.0433  0.0604  0.0449  0.0597  0.0355  0.0531  0.0343  0.1052
0.2120 ...  0.0083
 15  0.0298  0.0615  0.0650  0.0921  0.1615  0.2294  0.2176  0.2033  0.1459
0.0852 ...  0.0031
 16  0.0352  0.0116  0.0191  0.0469  0.0737  0.1185  0.1683  0.1541  0.1466
0.2912 ...  0.0346
 17  0.0192  0.0607  0.0378  0.0774  0.1388  0.0809  0.0568  0.0219  0.1037
0.1186 ...  0.0331
 18  0.0270  0.0092  0.0145  0.0278  0.0412  0.0757  0.1026  0.1138  0.0794
0.1520 ...  0.0084
 19  0.0126  0.0149  0.0641  0.1732  0.2565  0.2559  0.2947  0.4110  0.4983
0.5920 ...  0.0092

       52     53     54     55     56     57     58     59  60
 0   0.0065  0.0159  0.0072  0.0167  0.0180  0.0084  0.0090  0.0032   R
 1   0.0089  0.0048  0.0094  0.0191  0.0140  0.0049  0.0052  0.0044   R
 2   0.0166  0.0095  0.0180  0.0244  0.0316  0.0164  0.0095  0.0078   R
 3   0.0036  0.0150  0.0085  0.0073  0.0050  0.0044  0.0040  0.0117   R
 4   0.0054  0.0105  0.0110  0.0015  0.0072  0.0048  0.0107  0.0094   R
 5   0.0014  0.0038  0.0013  0.0089  0.0057  0.0027  0.0051  0.0062   R
 6   0.0248  0.0131  0.0070  0.0138  0.0092  0.0143  0.0036  0.0103   R
 7   0.0120  0.0045  0.0121  0.0097  0.0085  0.0047  0.0048  0.0053   R
 8   0.0128  0.0145  0.0058  0.0049  0.0065  0.0093  0.0059  0.0022   R
 9   0.0223  0.0179  0.0084  0.0068  0.0032  0.0035  0.0056  0.0040   R
 10  0.0120  0.0052  0.0056  0.0093  0.0042  0.0003  0.0053  0.0036   R
 11  0.0265  0.0224  0.0074  0.0118  0.0026  0.0092  0.0009  0.0044   R
 12  0.0127  0.0088  0.0098  0.0019  0.0059  0.0058  0.0059  0.0032   R
 13  0.0095  0.0194  0.0080  0.0152  0.0158  0.0053  0.0189  0.0102   R
 14  0.0057  0.0174  0.0188  0.0054  0.0114  0.0196  0.0147  0.0062   R
 15  0.0153  0.0071  0.0212  0.0076  0.0152  0.0049  0.0200  0.0073   R
 16  0.0158  0.0154  0.0109  0.0048  0.0095  0.0015  0.0073  0.0067   R
 17  0.0131  0.0120  0.0108  0.0024  0.0045  0.0037  0.0112  0.0075   R
 18  0.0010  0.0018  0.0068  0.0039  0.0120  0.0132  0.0070  0.0088   R
 19  0.0035  0.0098  0.0121  0.0006  0.0181  0.0094  0.0116  0.0063   R

[20 rows x 61 columns]
```

接下来看一下数据的描述性统计信息。代码如下：

```
# 描述性统计信息
set_option('precision', 3)
print(dataset.describe())
```

可以看到，数据具有相同的范围，但是中位值不同，这也许对数据正态化的结果有正面的影响。执行结果如下：

```
           0          1        2        3        4        5        6        7
8       9   \
 count   208.000   2.080e+02   208.000   208.000   208.000   208.000   208.000
208.000  208.000  208.000
 mean      0.029   3.844e-02     0.044     0.054     0.075     0.105     0.122
0.135    0.178    0.208
 std       0.023   3.296e-02     0.038     0.047     0.056     0.059     0.062
0.085    0.118    0.134
 min       0.002   6.000e-04     0.002     0.006     0.007     0.010     0.003
0.005    0.007    0.011
 25%       0.013   1.645e-02     0.019     0.024     0.038     0.067     0.081
0.080    0.097    0.111
 50%       0.023   3.080e-02     0.034     0.044     0.062     0.092     0.107
0.112    0.152    0.182
 75%       0.036   4.795e-02     0.058     0.065     0.100     0.134     0.154
0.170    0.233    0.269
 max       0.137   2.339e-01     0.306     0.426     0.401     0.382     0.373
0.459    0.683    0.711

         ...           50         51         52       53         54         55
56  \
 count     ...         208.000   2.080e+02   2.080e+02   208.000   2.080e+02
2.080e+02  2.080e+02
 mean      ...           0.016   1.342e-02   1.071e-02     0.011   9.290e-03
8.222e-03  7.820e-03
 std       ...           0.012   9.634e-03   7.060e-03     0.007   7.088e-03
5.736e-03  5.785e-03
 min       ...           0.000   8.000e-04   5.000e-04     0.001   6.000e-04
4.000e-04  3.000e-04
 25%       ...           0.008   7.275e-03   5.075e-03     0.005   4.150e-03
```

```
4.400e-03  3.700e-03
 50%        ...          0.014   1.140e-02   9.550e-03     0.009   7.500e-03
6.850e-03  5.950e-03
 75%        ...          0.021   1.673e-02   1.490e-02     0.015   1.210e-02
1.058e-02  1.043e-02
 max        ...          0.100   7.090e-02   3.900e-02     0.035   4.470e-02
3.940e-02  3.550e-02

             57        58        59
 count  2.080e+02  2.080e+02  2.080e+02
 mean   7.949e-03  7.941e-03  6.507e-03
 std    6.470e-03  6.181e-03  5.031e-03
 min    3.000e-04  1.000e-04  6.000e-04
 25%    3.600e-03  3.675e-03  3.100e-03
 50%    5.800e-03  6.400e-03  5.300e-03
 75%    1.035e-02  1.033e-02  8.525e-03
 max    4.400e-02  3.640e-02  4.390e-02

 [8 rows x 60 columns]
```

最后看一下数据分类的分布情况。代码如下：

```
 # 数据的分类分布
print(dataset.groupby(60).size())
```

由结果可以看到，两个数据分类大致是平衡的。结果如下：

```
 60
 M   111
 R    97
 dtype: int64
```

21.3.2 数据可视化

通常通过多种图表观察数据的分布情况，这会为解决问题提供灵感。我们先看一下每个数据特征的直方图。代码如下：

```
 # 直方图
dataset.hist(sharex=False, sharey=False,xlabelsize=1, ylabelsize=1)
pyplot.show()
```

从图 21-1 可以看到，大部分数据呈高斯分布或指数分布。

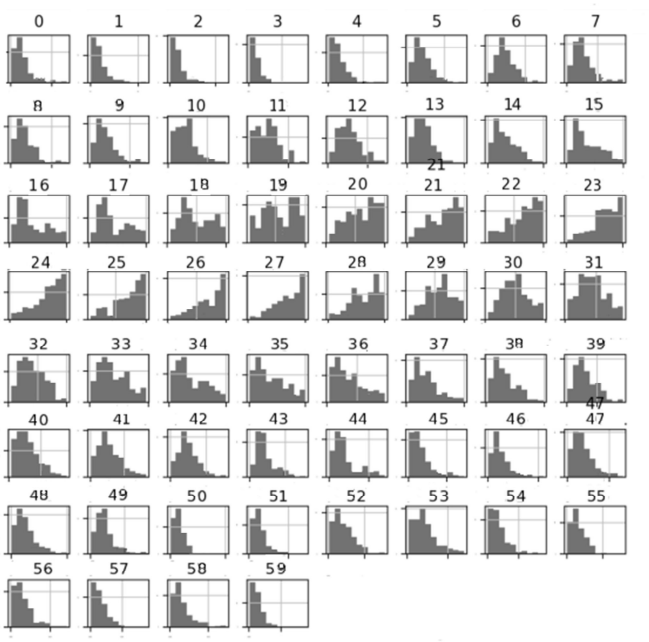


图 21-1

接下来看一下密度分布图。代码如下：

```
 # 密度图
dataset.plot(kind='density', subplots=True, layout=(8, 8), sharex=False,
legend=False, fontsize=1)
pyplot.show()
```

从图 21-2 可以看到，大部分数据都呈现一定程度的偏态分布，也许通过 Box-Cox 转换可以提高模型的准确度。Box-Cox 转换是统计中常用的一种数据变化方式，用于连续响应变量不满足正态分布的情况。Box-Cox 转换后，可以在一定程度上减少不可观测的误差，也可以预测变量的相关性，将数据转换成正态分布。

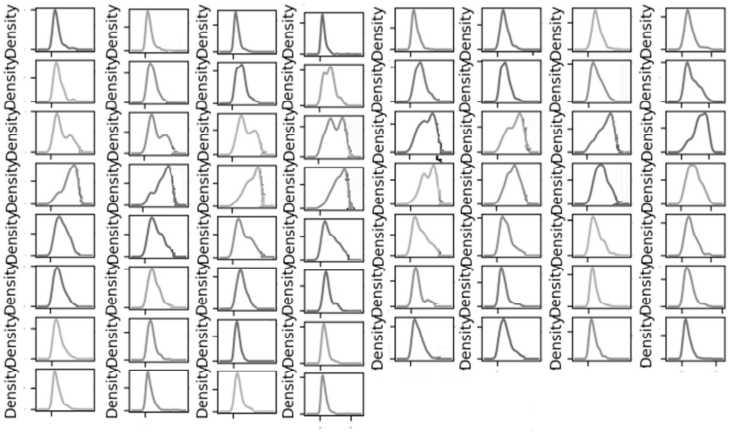


图 21-2

接下来看一下数据特征的两两相关性。代码如下：

```
 # 关系矩阵图
fig = pyplot.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(dataset.corr(), vmin=-1, vmax=1, interpolation='none')
fig.colorbar(cax)
pyplot.show()
```

可以看到数据有一定的负相关性，执行结果如图 21-3 所示。

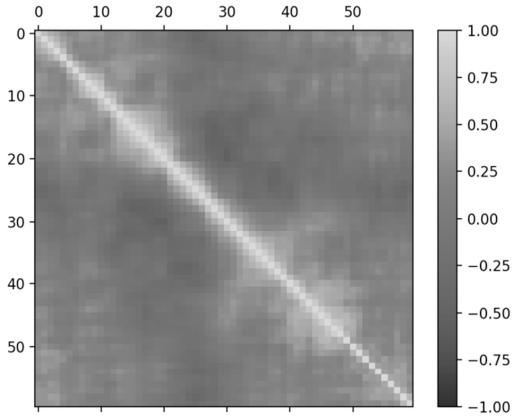


图 21-3

21.4 分离评估数据集

分离出一个评估数据集是一个很好的方法，可以确保分离出的数据集与训练模型的数据集完全隔离，这有助于最终的分析和判断模型的准确度 。在整个项目的最后，会使用这个评估数据集来确认模型的准确度，分离 20%的数据作为评估数据集，80%的数据作为训练数据集。代码如下：

```
 # 分离评估数据集
array = dataset.values
X = array[:, 0:60].astype(float)
Y = array[:, 60]
validation_size = 0.2
seed = 7
X_train, X_validation, Y_train, Y_validation =
train_test_split(X, Y, test_size=validation_size,
random_state=seed)
```

21.5 评估算法

分析完数据后，并不能决定选择哪个算法对这个问题最有效。直观的感觉是，也许依赖距离计算的算法会给出比较准确的结果，但仍然无法确定哪个算法会生产准确度最高的模型，因此需要设计一个评估框架来帮助我们选择合适的算法。在这里采用 10 折交叉验证来分离数据，并通过准确度来比较算法，这样可以很快地找到最优算法。代码如下：

```
 # 评估算法的基准
num_folds = 10
seed = 7
scoring = 'accuracy'
```

首先利用原始数据对算法进行审查，下面会选择六种不同的算法进行审查。

线性算法：逻辑回归算法（LR）和线性判别分析（LDA）。

非线性算法：分类与回归树算法（CART）、支持向量机（SVM）、贝叶斯分类器（NB）和 K 近邻算法（KNN）。

算法模型初始化的代码如下：

```
# 评估算法——原始数据
models = {}
models['LR'] = LogisticRegression()
models['LDA'] = LinearDiscriminantAnalysis()
models['KNN'] = KNeighborsClassifier()
models['CART'] = DecisionTreeClassifier()
models['NB'] = GaussianNB()
models['SVM'] = SVC()
```

对所有的算法都不进行调参，使用默认参数来比较算法。通过比较准确度的平均值和标准方差来比较算法。代码如下：

```
results = []
for key in models:
    kfold = KFold(n_splits=num_folds, random_state=seed)
    cv_results = cross_val_score(models[key], X_train, Y_train, cv=kfold,
scoring=scoring)
    results.append(cv_results)
    print('%s : %f (%f)' % (key, cv_results.mean(),
 cv_results.std()))
```

执行结果显示，逻辑回归算法（LR）和 K 近邻算法（KNN） 值得我们进一步进行分析和研究。执行结果如下：

```
LR : 0.782721 (0.093796)
LDA : 0.746324 (0.117854)
KNN : 0.808088 (0.067507)
CART : 0.723529 (0.093295)
NB : 0.648897 (0.141868)
SVM : 0.608824 (0.118656)
```

这只是 K 折交叉验证给出的平均统计结果，通常还要看算法每次得出的结果的分布状况。在这里使用箱线图来显示数据的分布状况。代码如下：

```
# 评估算法——箱线图
fig = pyplot.figure()
```

```
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(models.keys())
pyplot.show()
```

执行结果如图 21-4 所示，可以看到 K 近邻算法（KNN）的执行结果分布比较紧凑，说明算法对数据的处理比较准确。但是，支持向量机（SVM）算法的结果比较差，这出乎我们的意料。

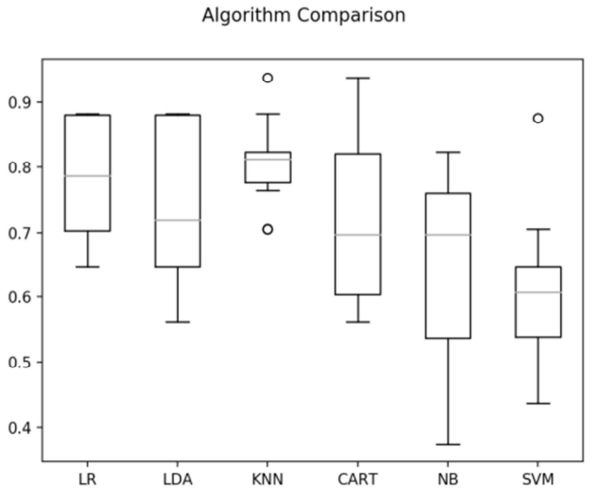


图 21-4

也许是因为数据分布的多样性导致支持向量机（SVM）算法不够准确，接下来会对数据进行正态化，然后重新评估算法。

我们猜想原始数据分布不均匀，导致有些算法表现不佳，于是将数据进行正态化处理，然后对算法再次进行评估。为了确保数据的一致性，将采用 Pipeline 来流程化处理。代码如下：

```
 # 评估算法——正态化数据
pipelines = {}
pipelines['ScalerLR'] = Pipeline([('Scaler',
StandardScaler()), ('LR', LogisticRegression())])
```

```
pipelines['ScalerLDA'] = Pipeline([('Scaler',
StandardScaler()), ('LDA', LinearDiscriminantAnalysis())])
pipelines['ScalerKNN'] = Pipeline([('Scaler',
StandardScaler()), ('KNN', KNeighborsClassifier())])
pipelines['ScalerCART'] = Pipeline([('Scaler',
StandardScaler()), ('CART', DecisionTreeClassifier())])
pipelines['ScalerNB'] = Pipeline([('Scaler',
StandardScaler()), ('NB', GaussianNB())])
pipelines['ScalerSVM'] = Pipeline([('Scaler',
StandardScaler()), ('SVM', SVC())])
results = []
for key in pipelines:
    kfold = KFold(n_splits=num_folds, random_state=seed)
    cv_results = cross_val_score(pipelines[key], X_train, Y_train, cv=kfold,
scoring=scoring)
    results.append(cv_results)
    print('%s : %f (%f)' % (key, cv_results.mean(), cv_results.std()))
```

从执行结果可以看到，K 近邻算法（KNN）依然具有最好的结果，甚至还有所提高，同时支持向量机算法（SVM）也得到了极大的提高。执行结果如下：

```
ScalerLR : 0.734191 (0.095885)
ScalerLDA : 0.746324 (0.117854)
ScalerKNN : 0.825735 (0.054511)
ScalerCART : 0.764706 (0.102797)
ScalerNB : 0.648897 (0.141868)
ScalerSVM : 0.836397 (0.088697)
```

再通过箱线图来看一下每次执行结果的分布情况。代码如下：

```
# 评估算法——箱线图
fig = pyplot.figure()
fig.suptitle('Scaled Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(models.keys())
pyplot.show()
```

执行结果如图 21-5 所示，同样可以看到 K 近邻算法（KNN）和支持向量机（SVM）的数据分布也是最紧凑的。

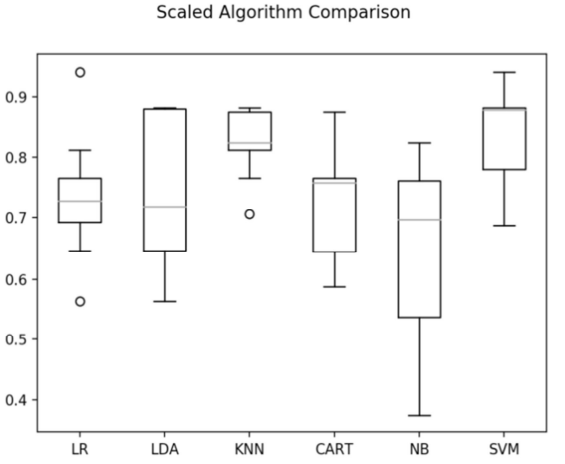


图 21-5

21.6 算法调参

通过对算法的评估，发现 K 近邻算法（KNN）和支持向量机（SVM）值得我们进一步进行优化，下面就对这两个算法进行调参，进一步提高算法的准确度。

21.6.1 K 近邻算法调参

K 近邻算法（KNN）默认的 n_neighbors 数是 5，下面将对 n_neighbors 数为 21 的奇数进行调参。同样采用 10 折交叉验证来确认最优参数。代码如下：

```
# 调参改进算法——KNN
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
param_grid = {'n_neighbors': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21]}
model = KNeighborsClassifier()
kfold = KFold(n_splits=num_folds, random_state=seed)
grid = GridSearchCV(estimator=model,
```

```
param_grid=param_grid, scoring=scoring, cv=kfold)
grid_result = grid.fit(X=rescaledX, y=Y_train)

print('最优：%s 使用%s' % (grid_result.best_score_, grid_result.best_params_))
cv_results =
zip(grid_result.cv_results_['mean_test_score'],

grid_result.cv_results_['std_test_score'],
                grid_result.cv_results_['params'])
for mean, std, param in cv_results:
   print('%f (%f) with %r' % (mean, std, param))
```

将所有的执行结果打印如下：

```
最优：0.849397590361 使用{'n_neighbors': 1}
0.849398 (0.059881) with {'n_neighbors': 1}
0.837349 (0.066303) with {'n_neighbors': 3}
0.837349 (0.037500) with {'n_neighbors': 5}
0.765060 (0.089510) with {'n_neighbors': 7}
0.753012 (0.086979) with {'n_neighbors': 9}
0.734940 (0.104890) with {'n_neighbors': 11}
0.734940 (0.105836) with {'n_neighbors': 13}
0.728916 (0.075873) with {'n_neighbors': 15}
0.710843 (0.078716) with {'n_neighbors': 17}
0.722892 (0.084555) with {'n_neighbors': 19}
0.710843 (0.108829) with {'n_neighbors': 21}
```

通过结果可以发现，最优的 n_neighbors 数是 1，也就是说，在预测数据时，需要寻找最接近的结果。

21.6.2 支持向量机调参

支持向量机有两个重要的参数，C（惩罚系数）和 kernel（径向基函数），默认的 C 参数是 1.0，默认的 kernel 参数是 rbf。下面将对这两个参数进行调参。代码如下：

```
# 调参改进算法——SVM
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train).astype(float)
```

```
param_grid = {}
param_grid['C'] = [0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 1.3, 1.5, 1.7, 2.0]
param_grid['kernel'] = ['linear', 'poly', 'rbf', 'sigmoid', 'precomputed']
model = SVC()
kfold = KFold(n_splits=num_folds, random_state=seed)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring,
cv=kfold)
grid_result = grid.fit(X=rescaledX, y=Y_train)

print('最优：%s 使用%s' % (grid_result.best_score_, grid_result.best_params_))
cv_results = zip(grid_result.cv_results_['mean_test_score'],
                 grid_result.cv_results_['std_test_score'],
                 grid_result.cv_results_['params'])
for mean, std, param in cv_results:
   print('%f (%f) with %r' % (mean, std, param))
```

执行之后得到如下结果：

```
最优：0.867469879518 使用{'C': 1.5, 'kernel': 'rbf'}
0.759036 (0.098863) with {'C': 0.1, 'kernel': 'linear'}
0.530120 (0.118780) with {'C': 0.1, 'kernel': 'poly'}
0.572289 (0.130339) with {'C': 0.1, 'kernel': 'rbf'}
0.704819 (0.066360) with {'C': 0.1, 'kernel': 'sigmoid'}
0.746988 (0.108913) with {'C': 0.3, 'kernel': 'linear'}
0.644578 (0.132290) with {'C': 0.3, 'kernel': 'poly'}
0.765060 (0.092312) with {'C': 0.3, 'kernel': 'rbf'}
0.734940 (0.054631) with {'C': 0.3, 'kernel': 'sigmoid'}
0.740964 (0.083035) with {'C': 0.5, 'kernel': 'linear'}
0.680723 (0.098638) with {'C': 0.5, 'kernel': 'poly'}
0.789157 (0.064316) with {'C': 0.5, 'kernel': 'rbf'}
0.746988 (0.059265) with {'C': 0.5, 'kernel': 'sigmoid'}
0.746988 (0.084525) with {'C': 0.7, 'kernel': 'linear'}
0.740964 (0.127960) with {'C': 0.7, 'kernel': 'poly'}
0.813253 (0.084886) with {'C': 0.7, 'kernel': 'rbf'}
0.753012 (0.058513) with {'C': 0.7, 'kernel': 'sigmoid'}
0.759036 (0.096940) with {'C': 0.9, 'kernel': 'linear'}
0.771084 (0.102127) with {'C': 0.9, 'kernel': 'poly'}
0.837349 (0.087854) with {'C': 0.9, 'kernel': 'rbf'}
0.753012 (0.073751) with {'C': 0.9, 'kernel': 'sigmoid'}
```

```
0.753012 (0.099230) with {'C': 1.0, 'kernel': 'linear'}
0.789157 (0.107601) with {'C': 1.0, 'kernel': 'poly'}
0.837349 (0.087854) with {'C': 1.0, 'kernel': 'rbf'}
0.753012 (0.070213) with {'C': 1.0, 'kernel': 'sigmoid'}
0.771084 (0.106063) with {'C': 1.3, 'kernel': 'linear'}
0.819277 (0.106414) with {'C': 1.3, 'kernel': 'poly'}
0.849398 (0.079990) with {'C': 1.3, 'kernel': 'rbf'}
0.710843 (0.076865) with {'C': 1.3, 'kernel': 'sigmoid'}
0.759036 (0.091777) with {'C': 1.5, 'kernel': 'linear'}
0.831325 (0.109499) with {'C': 1.5, 'kernel': 'poly'}
0.867470 (0.090883) with {'C': 1.5, 'kernel': 'rbf'}
0.740964 (0.063717) with {'C': 1.5, 'kernel': 'sigmoid'}
0.746988 (0.090228) with {'C': 1.7, 'kernel': 'linear'}
0.831325 (0.115695) with {'C': 1.7, 'kernel': 'poly'}
0.861446 (0.087691) with {'C': 1.7, 'kernel': 'rbf'}
0.710843 (0.088140) with {'C': 1.7, 'kernel': 'sigmoid'}
0.759036 (0.094276) with {'C': 2.0, 'kernel': 'linear'}
0.831325 (0.108279) with {'C': 2.0, 'kernel': 'poly'}
0.867470 (0.094701) with {'C': 2.0, 'kernel': 'rbf'}
0.728916 (0.095050) with {'C': 2.0, 'kernel': 'sigmoid'}
```

最好的支持向量机（SVM）的参数是 C=1.5 和 kernel=RBF。准确度能够达到 0.8675，这也比 K 近邻算法（KNN）的结果要好一些。

21.7 集成算法

除了调参，提高算法准确度的方法是集成算法。下面会对四种集成算法进行比较，以便进一步提高算法的准确度。

装袋算法：随机森林（RF）和极端随机树（ET）。

提升算法：AdaBoost（AB）和随机梯度上升（GBM）。

依然采用 10 折交叉验证来验证集成算法的准确度，以便选择最优的算法模型。代码如下：

```
# 集成算法
ensembles = {}
ensembles['ScaledAB'] = Pipeline([('Scaler',
StandardScaler()), ('AB', AdaBoostClassifier())])
ensembles['ScaledGBM'] = Pipeline([('Scaler',
StandardScaler()), ('GBM', GradientBoostingClassifier())])
ensembles['ScaledRF'] = Pipeline([('Scaler',
StandardScaler()), ('RFR', RandomForestClassifier())])
ensembles['ScaledET'] = Pipeline([('Scaler',
StandardScaler()), ('ETR', ExtraTreesClassifier())])

results = []
for key in ensembles:
    kfold = KFold(n_splits=num_folds, random_state=seed)
    cv_result = cross_val_score(ensembles[key], X_train, Y_train, cv=kfold,
scoring=scoring)
    results.append(cv_result)
    print('%s: %f (%f)' % (key, cv_result.mean(), cv_result.std()))
```

执行结果如下：

```
ScaledAB: 0.813971 (0.066017)
ScaledGBM: 0.848162 (0.106352)
ScaledRF: 0.745588 (0.108712)
ScaledET: 0.806618 (0.076435)
```

通过箱线图来看一下算法结果的离散状况。代码如下：

```
# 集成算法 ——箱线图
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(ensembles.keys())
pyplot.show()
```

执行结果如图 21-6 所示。

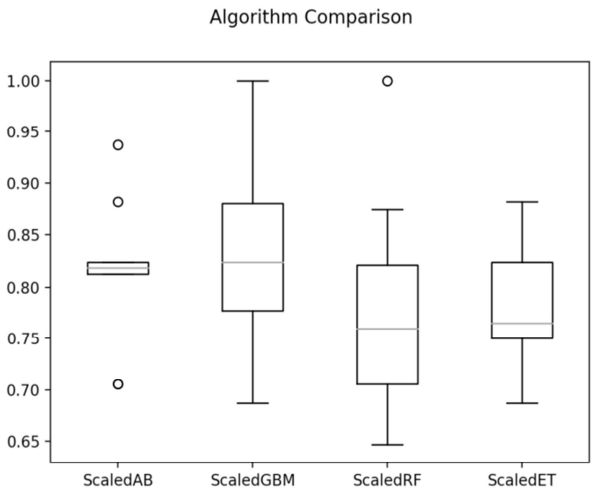


图 21-6

我们可以看到，随机梯度上升（GBM）也许值得进行进一步的分析，因为它具有良好的准确度，并且数据比较紧凑。接下来对其进行调参，代码如下：

```
# 集成算法 GBM——调参
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
param_grid = {'n_estimators': [10, 50, 100, 200, 300, 400, 500, 600, 700, 800,
900]}
model = GradientBoostingClassifier()
kfold = KFold(n_splits=num_folds, random_state=seed)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring,
cv=kfold)
grid_result = grid.fit(X=rescaledX, y=Y_train)
print('最优：%s 使用%s' % (grid_result.best_score_, grid_result.best_params_))
```

调参之后的结果如下：

```
最优：0.861445783133 使用{'n_estimators': 500}
```

21.8 确定最终模型

通过前面对算法的评估发现，支持向量机（SVM）具有最佳的准确度。所以将会采用支持向量机（SVM），通过训练集数据生成算法模型，并通过预留的评估数据集来评估模型。在算法评估过程中发现，支持向量机（SVM）对正态化的数据具有较高的准确度。所以对训练集做正态化处理，对评估数据集也做相同的处理，代码如下：

```
 # 模型最终化
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
model = SVC(C=1.5, kernel='rbf')
model.fit(X=rescaledX, y=Y_train)
# 评估模型
rescaled_validationX = scaler.transform(X_validation)
predictions = model.predict(rescaled_validationX)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

从执行结果可以看到，准确度大概达到了 86%，这是与期待结果比较接近的结果。执行结果如下：

```
0.857142857143
[[23  4]
 [ 2 13]]
             precision    recall  f1-score   support

          M       0.92      0.85      0.88       27
          R       0.76      0.87      0.81       15

avg / total       0.86      0.86      0.86       42
```

21.9 总结

通过本章的学习进一步了解了分类问题的项目流程，在本章学到的技巧可以直接应

用到实际的项目当中。到目前为止，机器学习的基本知识，以及分类与回归问题的处理方法，都可以直接应用在实践中。机器学习是跨领域的，且集合了多学科的知识，对于机器学习的掌握，需要通过大量的练习和不断地学习。下一章将会介绍分类问题的一个分支——文本分类问题，这也是实际工作中比较常见的类型，如垃圾邮件自动分类、新闻自动分类等问题。

22 文本分类实例

前面介绍了对数值问题的分类与回归问题的实例。在实际应用中还有一类问题比较重要，那就是文本分类。接下来将会通过一个例子介绍文本分类。

- 如何端到端地完成一个文本分类问题的模型。
- 如何通过文本特征提取生成数据特征。
- 如何通过调参提高模型的准确度。
- 如何通过集成算法提高模型的准确度。

22.1 问题定义

在这个项目中会采用 20 Newgroups 的数据(http://qwone.com/~jason/20Newsgroups/)，这是网上非常流行的对文本进行分类和聚类的数据集。数据集中的数据分为两部分，一部分是用来训练算法模型的数据，一部分是用来评估算法的新数据。网站上还提供了 3 个数据集，这里采用 20news-bydate 这个数据集进行项目研究。这个数据集是按照日期进行排序的，并去掉了部分重复数据和 header，共包含 18846 个文档。

22.2 导入数据

这里使用 scikit-learn 的 loadfiles 导入文档数据，文档是按照不同的分类分目录来保存的，文件目录名称即所属类别，文件目录结构如图 22-1 所示。

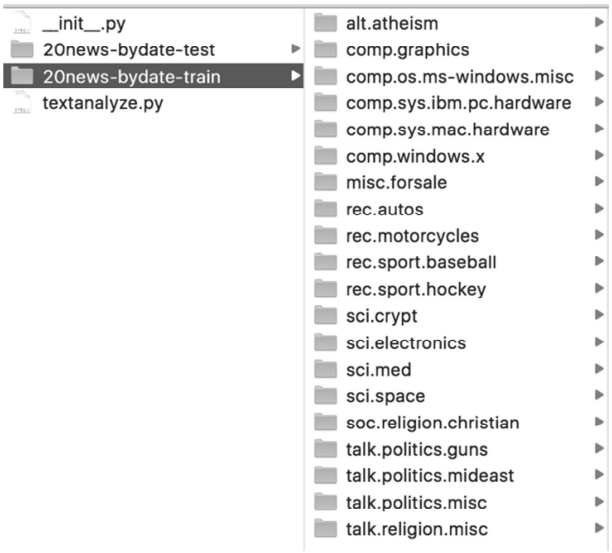


图 22-1

在导入文档数据之前，要导入项目中所需要的类库。代码如下：

```
from sklearn.datasets import load_files
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from matplotlib import pyplot as plt

# 导入数据
categories = ['alt.atheism',
              'rec.sport.hockey',
              'comp.graphics',
              'sci.crypt',
              'comp.os.ms-windows.misc',
              'sci.electronics',
              'comp.sys.ibm.pc.hardware',
              'sci.med',
              'comp.sys.mac.hardware',
              'sci.space',
              'comp.windows.x',
              'soc.religion.christian',
              'misc.forsale',
              'talk.politics.guns',
              'rec.autos'
              'talk.politics.mideast',
              'rec.motorcycles',
              'talk.politics.misc',
              'rec.sport.baseball',
              'talk.religion.misc']
# 导入训练数据
train_path = '20news-bydate-train'
dataset_train  = load_files(container_path=train_path,
categories=categories)
# 导入评估数据
test_path = '20news-bydate-test'
dataset_test = load_files(container_path=test_path,
categories=categories)
```

利用机器学习对文本进行分类，与对数值特征进行分类最大的区别是，对文本进行分类时要先提取文本特征，相对于之前的项目来说，提取到的文本特征属性的个数是巨大的，会有超过万个的特征属性，甚至会超过 10 万个。

22.3 文本特征提取

文本数据属于非结构化的数据，一般要转换成结构化的数据才能够通过机器学习算法进行文本分类。常见的做法是将文本转换成“文档-词项矩阵”，矩阵中的元素可以使用词频或 TF-IDF 值等。

TF-IDF 值是一种用于信息检索与数据挖掘的常用加权技术。TF 的意思是词频（Term Frequency），IDF 的意思是逆向文件频率（Inverse Document Frequency）。TF-IDF 的主要思想是：如果某一个词或短语在一篇文章中出现的频率高，并且在其他文章中很少出现，则认为此词或短语具有很好的类别区分能力，适合用来分类。TF-IDF 实际上是 TF * IDF。IDF 的主要思想是：如果包含词条 t 的文档越少，也就是 n 越小，IDF 越大，则说明词条 t 具有很好的类别区分能力。如果某一类文档 C 中包含词条 t 的文档数为 m，而其他类包含 t 的文档总数为 k，显然所有包含 t 的文档数 n=m+k，当 m 大的时候，n 也大，按照 IDF 公式得到的 IDF 的值小，这说明该词条 t 的类别区分能力不强。但是实际上，如果一个词条在一个类的文档中频繁出现，则说明该词条能够很好地代表这个类的文本特征，这样的词条应该被赋予较高的权重，并将其作为该类文本的特征词，以区别于其他类文档。这就是 IDF 的不足之处，在一份给定的文件里， TF 指的是某一个给定的词语在该文件中出现的频率，这是对词数（Term Count）的归一化，以防止它偏向长的文件。IDF 是一个词语普遍重要性的度量，某一个特定词语的 IDF，可以由总文件数目除以包含该词语的文件的数目，再将得到的商取对数得到。

在 scikit-learn 中提供了词频和 TF-IDF 来进行文本特征提取的实现，分别是 CountVectorizer 和 TfidfTransformer。下面对训练数据集分别进行词频和 TF-IDF 的计算。代码如下：

```
 # 数据准备与理解
 # 计算词频
count_vect = CountVectorizer(stop_words='english',
decode_error='ignore')
X_train_counts =
count_vect.fit_transform(dataset_train.data)
```

```
 # 查看数据维度
print(X_train_counts.shape)
```

词频的计算结果如下：

```
 (10156, 122402)
```

接下来计算一下 TF-IDF，代码如下：

```
 # 计算 TF-IDF
tf_transformer = TfidfVectorizer(stop_words='english',
decode_error='ignore')
X_train_counts_tf =
tf_transformer.fit_transform(dataset_train.data)
 # 查看数据维度
print(X_train_counts_tf.shape)
```

TF-IDF 的计算结果如下：

```
 (10156, 122402)
```

这里通过两种方法进行了文本特征的提取，并且查看了数据维度，得到的数据维度还是非常巨大的。在后续的项目中，将使用 TF-IDF 进行分类模型的训练。因为，TF-IDF 的数据维度巨大，并且自用提取的特征数据，进一步对数据进行分析的意义不大，因此只简单地查看数据维度的信息。接下来将进行算法评估。

22.4 评估算法

通过简单地查看数据维度，不能确定哪个算法对这个问题比较有效。下面将采用 10 折交叉验证的方式来比较算法的准确度，以便找到处理问题最有效的两三种算法，然后进行下一步的处理。代码如下：

```
 # 设置评估算法的基准
num_folds = 10
seed = 7
scoring = 'accuracy'
```

接下来将会利用提取到的文本特征 TF-IDF 来对算法进行审查，审查的算法如下。

线性算法：逻辑回归（LR）。

非线性算法：分类与回归树（CART）、支持向量机（SVM）、朴素贝叶斯分类器（MNB）和K近邻（KNN）。

算法模型的初始化代码如下：

```
# 评估算法
# 生成算法模型
models = {}
models['LR'] = LogisticRegression()
models['SVM'] = SVC()
models['CART'] = DecisionTreeClassifier()
models['MNB'] = MultinomialNB()
models['KNN'] = KNeighborsClassifier()
```

所有的算法使用默认参数，比较算法模型的准确度和标准方差，以便从中选择两三种可以进一步进行研究的算法。代码如下：

```
# 比较算法
results = []
for key in models:
    kfold = KFold(n_splits=num_folds, random_state=seed)
    cv_results = cross_val_score(models[key],
X_train_counts_tf, dataset_train.target, cv=kfold,
scoring=scoring)
    results.append(cv_results)
    print('%s : %f (%f)' % (key, cv_results.mean(),
cv_results.std()))
```

执行结果显示，逻辑回归（LR）具有最好的准确度，朴素贝叶斯分类器（MNB）和K近邻（KNN）也值得进一步的研究。执行结果如下：

```
LR : 0.901636 (0.010638)
SVM : 0.047658 (0.004797)
CART : 0.662171 (0.015739)
MNB : 0.880760 (0.006831)
KNN : 0.797163 (0.011413)
```

接下来看一下算法每次执行结果的分布情况。这里用箱线图来展示算法结果的分布情况。代码如下：

```
 # 箱线图比较算法
fig = plt.figure()
fig.suptitle('Algorithm Comparision')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(models.keys())
plt.show()
```

执行结果如图 22-2 所示，从图中可以看到，朴素贝叶斯分类器的数据离散程度比较好，逻辑回归的偏度较大。算法结果的离散程度能够反应算法对数据的适用情况，所以对逻辑回归和朴素贝叶斯分类器进行进一步的研究，实行算法调参。

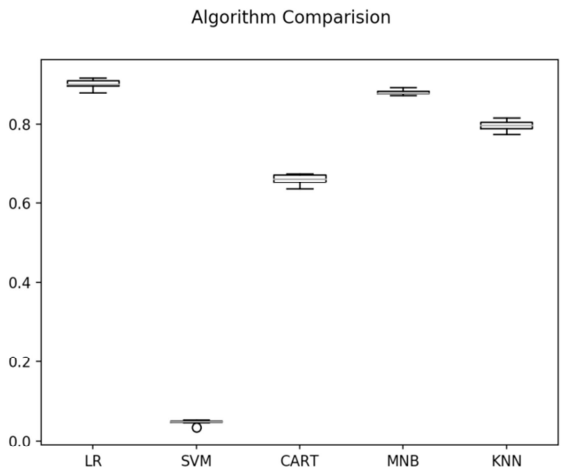


图 22-2

22.5 算法调参

通过上面的分析发现，逻辑回归（LR）和朴素贝叶斯分类器（MNB）算法值得进一步进行优化。下面对这两个算法的参数进行调参，进一步提高算法的准确度。

22.5.1 逻辑回归调参

在逻辑回归中的超参数是 *C*。*C* 是目标的约束函数，*C* 值越小则正则化强度越大。对 *C* 进行调参，每次给 *C* 设定一定数量的值，如果临界值是最优参数，重复这个步骤，直到找到最优值。代码如下：

```
# 算法调参
# 调参 LR
param_grid = {}
param_grid['C'] = [0.1, 5, 13, 15]
model = LogisticRegression()
kfold = KFold(n_splits=num_folds, random_state=seed)
grid = GridSearchCV(estimator=model,
param_grid=param_grid, scoring=scoring, cv=kfold)
grid_result = grid.fit(X=X_train_counts_tf,
y=dataset_train.target)
print('最优 : %s 使用 %s' % (grid_result.best_score_,
grid_result.best_params_))
```

可以看到 *C* 的最优参数是 13（13 是通过多次调整 param_grid 参数得到的）。执行结果如下：

```
最优 : 0.925167388736 使用 {'C': 13}
```

22.5.2 朴素贝叶斯分类器调参

通过对逻辑回归调参，准确度提升到大概 0.92，提升还是比较大的。朴素贝叶斯分类器有一个 alpha 参数，该参数是一个平滑参数，默认值为 1.0。我们也可以对这个参数进行调参，以提高算法的准确度。代码如下：

```
# 调参 MNB
param_grid = {}
param_grid['alpha'] = [0.001, 0.01, 0.1, 1.5]
model = MultinomialNB()
kfold = KFold(n_splits=num_folds, random_state=seed)
grid = GridSearchCV(estimator=model,
param_grid=param_grid, scoring=scoring, cv=kfold)
grid_result = grid.fit(X=X_train_counts_tf,
```

```
y=dataset_train.target)
print('最优 : %s 使用 %s' % (grid_result.best_score_, grid_result.best_params_))
```

同样，通过多次调整 param_grid，得到朴素贝叶斯分类器的 alpha 参数的最优值是 0.01。执行结果如下：

```
最优 : 0.916502560063 使用 {'alpha': 0.01}
```

通过调参发现，逻辑回归在 *C*=13 时具有最好的准确度。接下来审查集成算法。

22.6 集成算法

除了调参，提高算法准确度的方法是使用集成算法。下面对以下两种集成算法进行比较，看看能否进一步提高模型的准确度。

- 随机森林（RF）。
- AdaBoost（AB）。

集成算法的评估代码如下：

```
# 集成算法
ensembles = {}
ensembles['RF'] = RandomForestClassifier()
ensembles['AB'] = AdaBoostClassifier()
# 比较集成算法
results = []
for key in ensembles:
    kfold = KFold(n_splits=num_folds, random_state=seed)
    cv_results = cross_val_score(ensembles[key],
X_train_counts_tf, dataset_train.target, cv=kfold,
scoring=scoring)
    results.append(cv_results)
    print('%s : %f (%f)' % (key, cv_results.mean(), cv_results.std()))
```

对这两个算法的评估结果如下：

```
RF : 0.733162 (0.015826)
AB : 0.558684 (0.011587)
```

执行结果如图 22-3 所示，从箱线图可以看到随机森林（RF）的分布比较均匀，对数据的适用性比较高，更值得进一步优化研究。

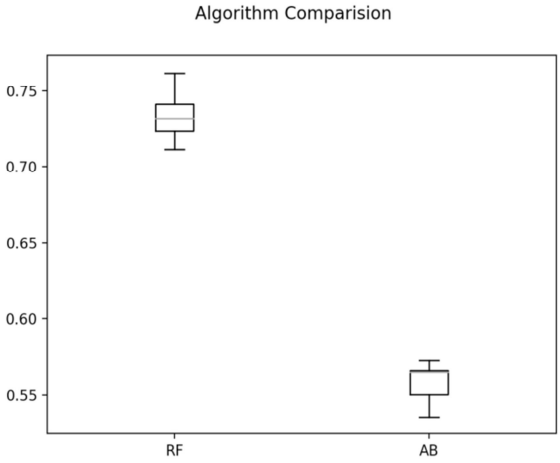


图 22-3

22.7 集成算法调参

通过对集成算法的分析，发现随机森林算法具有较高的准确度和非常稳定的数据分布，非常值得进行进一步的研究。下面通过调参对随机森林算法进行优化。随机森林有一个很重要的参数 n_estimators，下面对 n_estimators 进行调参优化，争取找到最优解。代码如下：

```
 # 调参 RF
param_grid = {}
param_grid['n_estimators'] = [10, 100, 150, 200]
model = RandomForestClassifier()
kfold = KFold(n_splits=num_folds, random_state=seed)
grid = GridSearchCV(estimator=model,
param_grid=param_grid, scoring=scoring, cv=kfold)
grid_result = grid.fit(X=X_train_counts_tf, y=dataset_train.target)
print('最优 : %s 使用 %s' % (grid_result.best_score_, grid_result.best_params_))
```

调参之后的最优结果如下：

```
最优 : 0.867467506892 使用 {'n_estimators': 150}
```

22.8 确定最终模型

通过对算法的比较和调参发现，逻辑回归算法具有最高的准确度，因此使用逻辑回归算法生成算法模型。接下来会利用评估数据集对生成的模型进行验证，以确认模型的准确度。需要注意的是，为了保持数据特征的一致性，对新数据进行文本特征提取时应进行特征扩充，下面使用之前生成的 tf_transformer 的 transform 方法来处理评估数据集。代码如下：

```
# 生成模型
model = LogisticRegression(C=13)
model.fit(X_train_counts_tf, dataset_train.target)
X_test_counts = tf_transformer.transform(dataset_test.data)
predictions = model.predict(X_test_counts)
print(accuracy_score(dataset_test.target, predictions))
print(classification_report(dataset_test.target,
predictions))
```

从结果可以看到，准确度大概达到了 84%，与期待的结果比较一致。执行结果如下：

```
0.844822485207
             precision    recall  f1-score   support

          0       0.84      0.77      0.80       319
          1       0.74      0.80      0.77       389
          2       0.77      0.74      0.76       394
          3       0.71      0.74      0.73       392
          4       0.81      0.85      0.83       385
          5       0.86      0.77      0.81       395
          6       0.83      0.91      0.87       390
          7       0.97      0.97      0.97       398
          8       0.91      0.94      0.92       397
          9       0.96      0.97      0.96       399
         10       0.96      0.93      0.94       396
```

```
         11       0.79      0.79      0.79       393
         12       0.91      0.88      0.89       396
         13       0.90      0.92      0.91       394
         14       0.86      0.94      0.89       398
         15       0.75      0.92      0.83       364
         16       0.87      0.61      0.71       310
         17       0.75      0.60      0.66       251

avg / total     0.85      0.84      0.84      6760
```

22.9 总结

本章主要介绍了对文本的分类方法，这类问题可以应用在垃圾邮件自动分类、新闻分类等方面。在文本分类中很重要的一点是文本特征提取，在进行文本特征提取时可以进一步优化，以提高模型的准确度。另外，对中文的文本分类，需要先进行分词，然后利用 sklearn.datasets.base.Bunch 将分词之后的文件加载到 scikit-learn 中。在运用机器学习处理问题时，需要灵活运用已经掌握的知识，寻找问题的解决方案。

附录 A

A.1 IDE PyCharm 介绍

PyCharm 是由 JetBrains 打造的一款 Python IDE，带有一整套可以帮助用户在使用 Python 语言进行开发时提高效率的工具，比如调试、语法高亮、Project 管理、代码跳转、智能提示、自动完成、单元测试、版本控制。此外，该 IDE 还提供了一些高级功能，以用于支持 Django 框架下的专业 Web 开发。

PyCharm 目前有两个版本：收费的专业版和免费的社区版，如图 A-1 所示，用户可以到 PyCharm 的官网自行下载（http://www.jetbrains.com/pycharm/download/）。专业版对利用 Python 语言进行 Web 开发提供了更多的支持，而社区版为平常的学习开发提供了足够的功能。本书中提供的示例代码全部使用 PyCharm 进行开发。

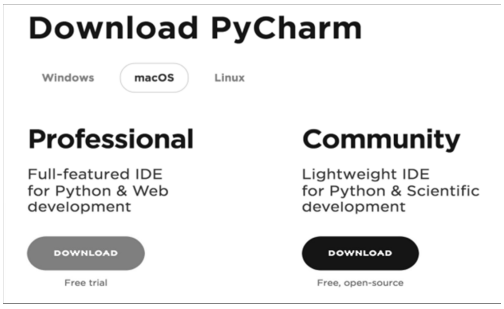


图 A-1

A.2 Python 文档

Python 官网提供了丰富的文档（https://www.python.org/doc/），涵盖从初级到高级的全部内容，所有的读者都可以找到适合自己的文档，如图 A-2 所示。

图 A-2

A.3 SciPy、NumPy、Matplotlib 和 Pandas 文档

SciPy、NumPy、Matplotlib 和 Pandas 的官网提供了丰富的文档，对 API 的适用有详细的使用说明。网址如下：

- SciPy——https://www.scipy.org/。
- NumPy——http://www.numpy.org/。
- Matplotlib——http://matplotlib.org/。
- Pandas——http://pandas.pydata.org/。

A.4 树模型可视化

安装 graphviz

在 Mac 上 graphviz 的安装是通过 MacPorts 进行的 ，这样可以自动安装 graphviz 的

所有依赖包。安装 MacPorts 时可以到官网（https://www.macports.org/）下载安装文件，直接安装即可。由于 graphviz 依赖 xcode 的一些库，需要先安装 xcode。安装 MacPorts 之后，执行以下命令完成对 graphviz 的安装。

```
sudo port install graphviz-gui
```

决策树结果可视化

使用鸢尾花数据集演示如何将决策树的结果可视化，数据集可以直接到 UCI 机器学习仓库（http://archive.ics.uci.edu/ml/datasets/Iris）下载。代码如下：

```
from pandas import read_csv
from matplotlib.image import imread
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
from sklearn.metrics import accuracy_score
import pydotplus
import os

# 导入数据
filename = 'iris.data.csv'
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width',
'class']
dataset = read_csv(filename, names=names)
array = dataset.values
X = array[:,0:4]
y = array[:,4]

# 分离数据集
X_train, X_test, y_train, y_test = train_test_split(X,
y, random_state=7, test_size=0.2)

# 决策树模型训练
model = DecisionTreeClassifier()
model.fit(X=X_train, y=y_train)
# 决策树图形化
dot_data = export_graphviz(model, out_file=None)
graph = pydotplus.graph_from_dot_data(dot_data)
path = os.getcwd() + '/'
```

```
tree_file = path + 'iris.png'
try:
    os.remove(tree_file)
except:
    print('There is no file to be deleted.')
finally:
    graph.write(tree_file, format='png')

#显示图像
image_data = imread(tree_file)
pyplot.imshow(image_data)
pyplot.axis('off')
pyplot.show()

# 评估算法
predictions = model.predict(X_test)
print(accuracy_score(y_test, predictions))
```

生成决策树的算法模型图像时，首先利用 scikit-learn 的方法导出 dot 描述，并通过 dot 来生成图像。通过生成的图像，可以很方便地看到在树模型中如何进行分类。执行上述代码生成的决策树模型如图 A-3 所示。

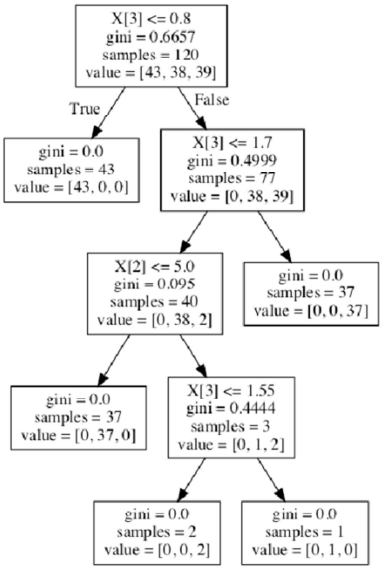


图 A-3

A.5 scikit-learn 的算法选择路径

图 A-4 是 scikit-learn 官网的算法选择路径图的英文翻译，在选择算法时可以将其作为参考。

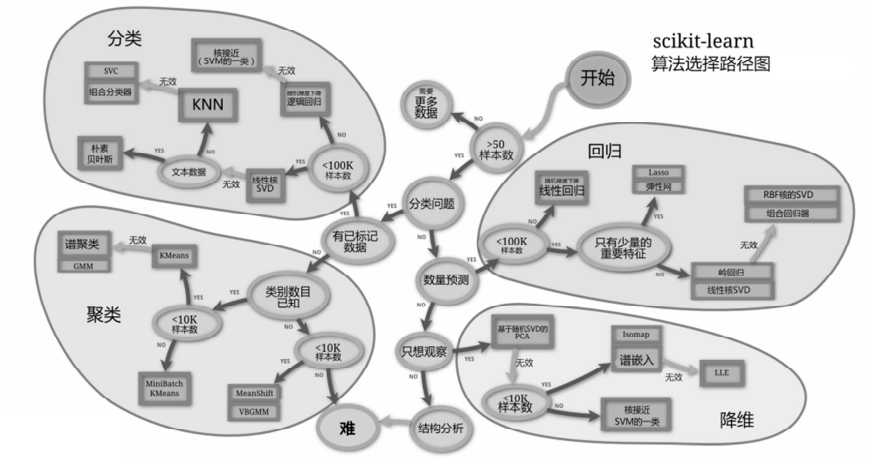


图 A-4

A.6 聚类分析

聚类（Cluster）分析又称群分析，是研究分类（样品或指标）问题的一种统计分析方法，也是数据挖掘的一个重要算法。聚类分析是由若干模式（Pattern）组成的，通常，模式是一个度量（Measurement）的向量，或者是多维空间中的一个点。聚类分析以相似性为基础，在同一个聚类中的模式之间比不在同一个聚类中的模式之间具有更多的相似性。

许多聚类算法在小于 200 个数据对象的小数据集合上工作得很好；但是，一个大规模的数据库可能包含几百万个对象，在这样的大数据集合样本上进行聚类可能会导致有偏差的结果。因此需要具有高度可伸缩性的聚类算法。

许多聚类算法擅长处理低维的数据，可能只涉及两维或三维。人类的眼睛在最多三

维的情况下能够很好地判断聚类的质量。在高维空间中，聚类数据对象是非常有挑战性的，特别是考虑到这样的数据可能分布非常稀疏，而且高度偏斜。因此，对于多维度的数据进行聚类算法时首先要进行降维处理。

接下来通过一个例子来展示聚类算法的应用。在这里使用 UCI 数据仓库中的 wine 数据集（http://archive.ics.uci.edu/ml/datasets/Wine），这个数据集中包含 13 个数据特征，并且数据被分为三个类别，通过 KMean 算法自动聚类。代码如下：

```
from pandas import read_csv
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
from sklearn.preprocessing import StandardScaler
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
from sklearn import metrics

# 导入数据
filename = 'wine.data'
names = ['class', 'Alcohol', 'MalicAcid', 'Ash',
'AlclinityOfAsh', 'Magnesium', 'TotalPhenols',
'Flavanoids',
       'NonflayanoidPhenols', 'Proanthocyanins',
'ColorIntensiyt', 'Hue', 'OD280/OD315', 'Proline']
dataset = read_csv(filename, names=names)
dataset['class'] =
dataset['class'].replace(to_replace=[1, 2, 3], value=[0, 1, 2])
array = dataset.values
X = array[:, 1:13]
y = array[:, 0]

# 数据降维
pca = PCA(n_components=3)
X_scale = StandardScaler().fit_transform(X)
X_reduce = pca.fit_transform(scale(X_scale))

# 模型训练
```

```
model = KMeans(n_clusters=3)
model.fit(X_reduce)
labels = model.labels_
#print(labels)

# 输出模型的准确度
print('%.3f   %.3f   %.3f   %.3f   %.3f    %.3f' %(
    metrics.homogeneity_score(y, labels),
    metrics.completeness_score(y, labels),
    metrics.v_measure_score(y, labels),
    metrics.adjusted_rand_score(y, labels),
    metrics.adjusted_mutual_info_score(y,  labels),
    metrics.silhouette_score(X_reduce, labels)))

# 绘制模型的分布图
fig = plt.figure()
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(X_reduce[:, 0], X_reduce[:, 1], X_reduce[:,
2], c=labels.astype(np.float))
plt.show()
```

这里仅简单演示聚类的实现方式，代码不做详细讲解，聚类结果如图 A-5 所示，执行结果如下：

```
 0.726   0.722   0.724   0.726   0.719    0.413
```

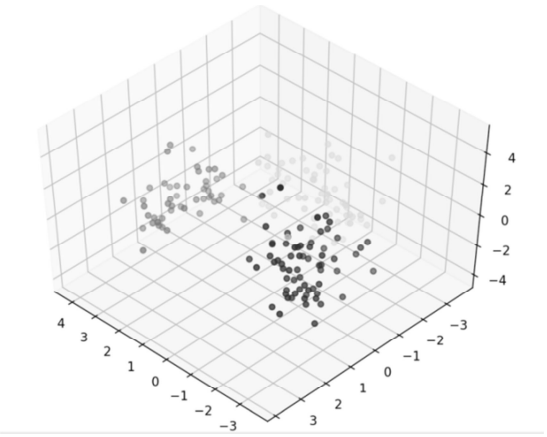


图 A-5

反侵权盗版声明

电子工业出版社依法对本作品享有专有出版权。任何未经权利人书面许可，复制、销售或通过信息网络传播本作品的行为；歪曲、篡改、剽窃本作品的行为，均违反《中华人民共和国著作权法》，其行为人应承担相应的民事责任和行政责任，构成犯罪的，将被依法追究刑事责任。

为了维护市场秩序，保护权利人的合法权益，我社将依法查处和打击侵权盗版的单位和个人。欢迎社会各界人士积极举报侵权盗版行为，本社将奖励举报有功人员，并保证举报人的信息不被泄露。

举报电话：（010）88254396；（010）88258888

传　　真：（010）88254397

E - m a i l：dbqq@phei.com.cn

通信地址：北京市万寿路 173 信箱

　　电子工业出版社总编办公室

邮　　编：100036